Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Load management for a telecom charging system

by

Johan Bjerre

LIU-IDA/LITH-EX-A--08/043--SE

2008-10-13

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Final Thesis

Load management for a telecom charging system

by

Johan Bjerre

LIU-IDA/LITH-EX-A--08/043--SE

2008-10-13

Supervisor: Jim Håkansson

Examinator: Simin Nadjm-Tehrani

Avdelning, Division

Department of Computer
and Information Science

Datum, date
2008-10-13

Linköpings universitet

Rapporttyp, Report category

	Licentiatavhandling
X	Examensarbete
	C-uppsats
	D-uppsats

Språk, language

	Svenska/Swedish
X	Engelska/English

ISBN
-
ISRN
LIU-IDA/LITH-EX-A--08/043--SE

Serietitel och serienummer, ISSN
Title of series, numbering
-

URL

Titel, title
Load management for a telecom charging system

Författare, author
Johan Bjerre

Abstrakt, abstract
There are a huge number of customers using Ericsson's prepaid telecom charging system. This means that even the smallest optimization done in its management of load results in a big win when considering time and CPU usage. Ericsson wishes therefore to learn about how the load management can be improved considering speed, i.e. faster response to load changes.

In this thesis work two subsystems of the prepaid telecom system are modeled. The first subsystem treats and prepares the telecom activity before it enters the second subsystem which checks if there is coverage for the action that the telecom activity asks for to be performed.
A model which is an emulation of the subsystems is built using c++. It captures the timing of the real system not the logic or function. The c++ approach is compared to a Matlab approach which runs much slower in simulation. In addition, the model enables full view of the queue sizes and the rejects made by the regulator.

Verification has been performed with self generated data based on synthetic test cases. Verification and analysis show that the model captures the intended behavior of load management. Information about different software parameters and its influence on the output is obtained. Different scenarios are tested and the outcome discussed.

Keywords
Telecom modelling, real time, time scheduling, load management

Abstract

There are a huge number of customers using Ericsson's prepaid telecom charging system. This means that even the smallest optimization done in its management of load results in a big win when considering time and CPU usage. Ericsson wishes therefore to learn about how the load management can be improved considering speed, i.e. faster response to load changes.

In this thesis work two subsystems of the prepaid telecom system are modeled. The first subsystem treats and prepares the telecom activity before it enters the second subsystem which checks if there is coverage for the action that the telecom activity asks for to be performed.
A model which is an emulation of the subsystems is built using c++. It captures the timing of the real system not the logic or function. The c++ approach is compared to a Matlab approach which runs much slower in simulation. In addition, the model enables full view of the queue sizes and the rejects made by the regulator.

Verification has been performed with self generated data based on synthetic test cases. Verification and analysis show that the model captures the intended behavior of load management. Information about different software parameters and its influence on the output is obtained. Different scenarios are tested and the outcome discussed.

Forewords

The following people have been helpful during the thesis:
Simin Nadjm-Tehrani
Jim Håkansson

Content

1 Introduction

In this chapter a background for the thesis is presented. The problem and the objective with the work are described. Also a short overview of the report is given.

1.1 Background

Most telecom activities made by an outside user are charged in some way. The user can mainly decide between two options; a subscription with a regular fee or paying all in advance, i.e. prepaid. The number of telecom users is increasing every day especially in fast expanding areas in Asia. The number of prepaid customers is also increasing compared to the customers with subscription. Prepaid has become popular since its clear control of the money flow.

When using telecom in an irregular way there is an advantage of prepaid since there are no fixed fees. In developing areas in Africa a common business is people who are making money out of selling calls. They charge a person for each call and that makes it possible for people in smaller villages to make calls without any big financial suffering. The post service might not be very developed and there might be difficulties in sending invoices.

Since Ericsson is one of the biggest suppliers of prepaid telecom charging systems, this means a huge number of customers. Many customers connecting to a system implies high demands on the performance of the systems handling the activities made by its users.

1.2 Description of the problem

This thesis addresses testing performance properties of a telecom charging service implemented at Ericsson. A model needs to be built to avoid a solution with real hard- and software. The hardware is expensive and it takes a lot of time to do simulations on a real system compared to a model with some smaller simplifications. The software needs to be installed on different kinds of hardware to investigate the influence of the hardware on the output.

Creating and setting up new tests is a complicated process and the company benefits from simplifying the process.

1.2.1 Output

There are limitations in the output from the simulator used by Ericsson. The information given by the measurements that are done are insufficient for analyzing the behavior of the telecom system. This thesis wishes therefore to expand the base of information on which the discussion about the behavior of the telecom system is done. The number of requests waiting in the different queues, the latencies of the requests and the number of rejects made by the load regulator are examples of measurement that are in need of expanding.

1.2.2 Input

There are also limitations in the input used by the simulator. The data generator is limited in a way that it cannot change the input very often. This causes a difficulty in generating a smooth increase or decrease in input.

1.3 Objective

This thesis wishes to enable the ability to analyze the telecom system in a more precise way than before. The model of the telecom system shall be able to handle the kind of input data which the telecom system is unable to handle. The increased ability to handle input data will enable more real life scenarios to be tested. These test cases will tell us how the telecom system would react in the same situations.

The model shall also give information about different software and hardware parameters and its influence on the output. These parameters, for example the number of threads and CPUs, shall be able to be changed easily.

1.4 Method

The main idea behind the work is presented in figure 1.1. Theoretical studies of the telecom charging system gave the material needed to construct a model. The model was verified against the telecom charging system. The verified model was then used for tests and analysis, which the conclusions were based on.

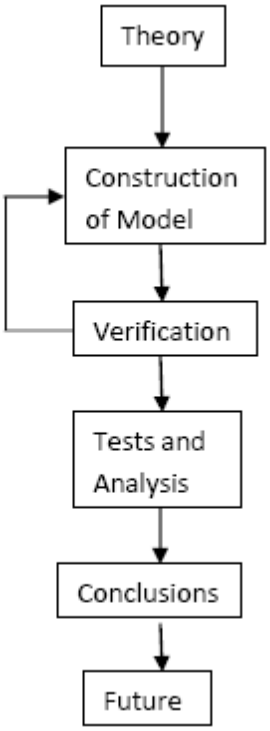


Figure 1.1: Method

1.5 Report Overview

Below follows a short description of the chapters

1. Introduction
   The problem together with the objective that is wished to be achieved is discussed. An overview of the method used is given as well.
2. Description of charging system
   The telecom charging system that the model is based on is discussed

3. Modeling choices
   Different approaches when it comes to creating the model are discussed. Choices made to capture the behavior of the telecom system are described.
4. Verification of model
   This chapter describes the methods used to verify the model against the telecom system.
5. Result
   In this chapter there is a discussion around the tests and analysis that has been done on the model.
6. Conclusions
   The knowledge achieved when testing and analyzing the model is presented.

   Appendix A, Glossary
   An example is given on the influence of the Poisson distribution on the latencies.

   Appendix B, Poisson distribution
   A summarizing datasheet comparing the different subsystems is given.

   Appendix C, Simulations
   Some of the words and expressions used in this thesis are explained further.

   Appendix D, Specifications
   All the tests that have been done are introduced and their outcome presented.

2 Description of charging system

This chapter gives an overview of the parts of the charging system that the thesis is about. The different parts and the system configuration behind are discussed.

2.1 Overview

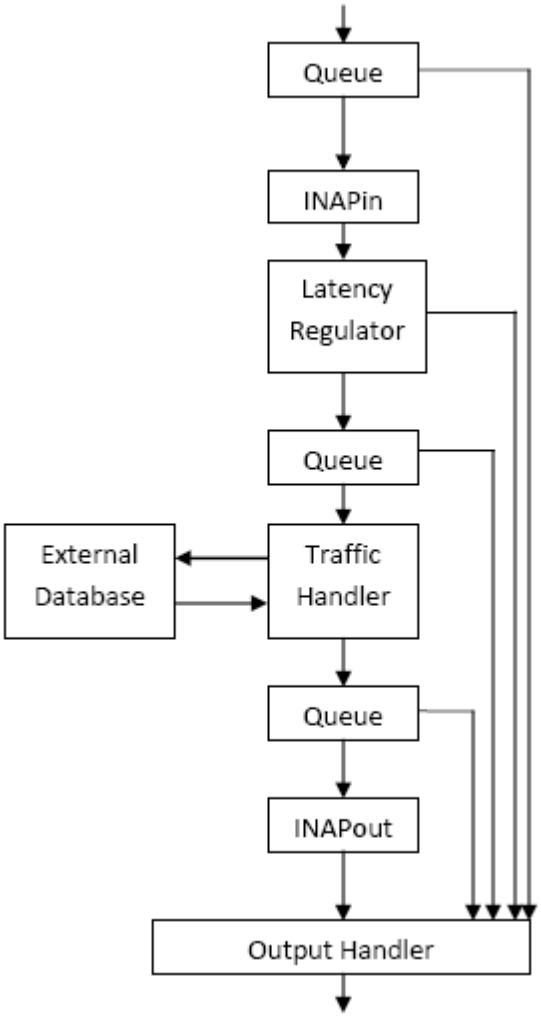


Figure 2.1: Overview of the telecom charging system

The part of the system that I have focused on is a subsystem handling the incoming requests to a prepaid telecom system and consists of mainly two subsystems which are called INAP (Intelligent Network Application Part) and TH (Traffic Handler). INAP is used twice for every request, once entering and once leaving the telecom system. It is therefore divided into two parts to give a better overview of the data flow in figure 2.1.

2.2 Queues

There are three queues placed before the subsystems (INAPin, INAPout and TH). The queues mission is to enable the ability to handle temporary overload. The requests will stay in the queue and wait until the next step in the flow are able to handle the request. When a request is waiting in a queue its latency will increase, therefore there are a maximum number of allowed requests waiting at the same time. This maximum has been set to avoid extremely long latencies. The queue before TH holds a maximum of 800 requests.

2.3 INAP and TH

INAP treats the requests and prepares them for the Traffic handler. The Traffic handler needs some specific parameters to be set to be able to handle the request. INAP therefore converts the request to be suitable for TH and when TH has finished its work, INAP converts the request back to how it was when it entered the telecom system.
TH checks if there is covering for the action that the telecom activity asks for. If there is covering, TH will charge the user for the activity carried out. A cost function will calculate how much the activity costs and charge the user. The cost function takes arguments like time elapsed, current price, data transferred and activity.

2.4 Traffic

Each request that enters the system has its own properties. A request can be from a SMS, phone call or perhaps some kind of purchase and may differ a lot in time spent in INAP and TH.
Some requests need access to external databases, this kind of external calls might cause requests to wait for a long time and get long latencies. The use of external databases causes the telecom system which has been studied, to be dependent on other systems. It might therefore now and then be difficult to explain its behavior.
Every SMS or phone call made is associated with a session, every session includes two requests. The first request includes checking the balance. The user needs to have a certain amount of money to be allowed to begin the activity. The second one charges the user.

2.4.1 Priority

Different request may have different priority. How to prioritize the different kinds of sessions differs between different companies. Most companies have a common interest in earning as much money as possible; therefore there is a higher priority on the second request in every session. This is since the second request is the one charging the user.

2.5 Output handler

Every request is time stamped when it enters the first queue in the telecom system. When the request reaches the Output handler its latency can be calculated. The latency of a request is the time taken from the moment the request enters the first queue until it, in a finished state, enters the Output handler. The response time is the time taken from entering the first queue until answered.

Decisions in the load management are based on the average latency which is updated with a specific interval of time.

2.6 Regulation

The customers using the charging system have demands on response time and therefore it is important to answer the request made by the customer. The request is answered when finished or rejected.
With very big queues there would be possible to work at 100% of the capacity at all time which may be considered optimal, but in that case requests will be waiting in the queues for a very long time which will result in large response times.
The latency regulation is meant to minimize the response time and give lower latencies. When having a heavy load you can sometime already from the start tell that the request will not be able to pass with an accepted latency. Therefore a certain percent of the requests is rejected by the latency regulator and answered. The request that are not rejected, are accepted and sent further.

2.6.1 Queue method

If a queue gets filled then the request with the lowest priority or the oldest request that is waiting in the queue will be answered and replaced as soon as another request enters the queue.
There is also a decision made whether the telecom system has a chance of being able of finishing the request. Therefore, if a request has already been in the queue for a long time, it might be answered even though the queue is not filled.

2.6.2 Latency time method

The latency time method is using an integrator, which is increased or decreased dependent on the average latency of the requests. The integrator works like this: The current average latency is compared to two levels. If the average latency is above the upper level then the integrator will decrease its level and if the average latency is below the lower level then the integrator will increase its level (figure 2.2). Latencies between the two levels are considered to be of no need of adjustment. The integrator has a maximum of 100%, i.e. all requests will be accepted and a minimum at a specific percentage to avoid all traffic to get rejected. The percentage of requests being rejected, i.e. the integrator level, is updated on every new latency average value.

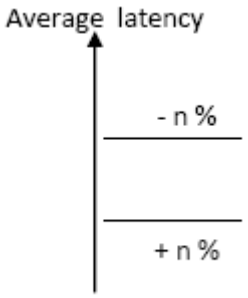


Figure 2.2: Latency time method

A request that is rejected according to the queue method instead of latency time method has already been delayed some time and therefore it is considered better to use the latency time method since it will result in a shorter response time.

2.7 System configuration

The system configuration is dependent on which the customer is and the performance needed. The case that has been studied has 4 CPU and a maximum of 24 threads.

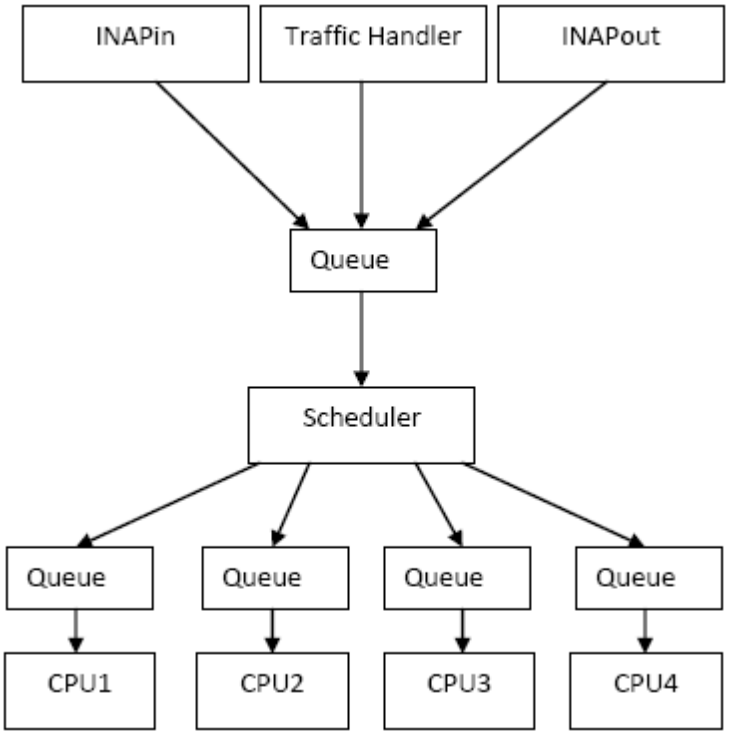


Figure 2.3: Scheduling

2.7.1 CPU

Every CPU can be considered as a working unit. Two CPU are capable to do twice as much work simultaneously as 1 CPU on its own. Adding a third would mean an extra 50% compared to two CPU. Two different CPUs cannot work on the same request. That means, when only having one request, having several CPUs does not give an increased speed. The increased speed caused by increased number of CPUs can therefore only be seen when the system has many incoming requests and there in no waiting, i.e. there is always a number of requests waiting in the queue.

2.7.2 Thread

Each thread may be considered as a working slot. Many active threads mean that many requests are processes in parallel. The number of active threads depends on how many requests that is being handled simultaneously. To be able to fully use the capacity of the system, there must be more threads than there are CPUs, since every working unit needs a working slot.

There is a maximum of 24 active threads distributed as 4:16:4 between INAPin, TH and INAPout. Each subsystem in the telecom system tries to use as many threads as possible out of the maximum 24. The first request, i.e. the one that has waited the longest in the queue will be allotted a thread as soon as any of the threads has finished its previous request.

2.7.3 Scheduling

All the requests arrive at a central processor, the scheduler in figure 2.3. The scheduler distributes the requests to the other four processors. The scheduler is running in parallel with CPU1-4 and makes sure that the CPUs never needs to wait for another request unless there is not enough incoming requests at the moment.

The CPUs are handling all the threads and then starts over. This is shown in figure 2.4. The threads are divided as 4:16:4, i.e. a maximum of 4 threads in INAPin (blue area in figure 2.4), maximum 16 threads in TH (red area) and 4 threads in INAPout (green area). This means that there are twice as many threads in the Traffic Handler than in INAP.

The black boxes in figure 2.4 indicate that the thread is active and enabled for processing of an eventual request. There are 4 CPUs, i.e. 4 working units and therefore a maximum of 4 threads can be processed at every tick. A tick is a moment of time happening every 10^{th} us.

If the thread has been allotted a request, then it will be processed when the thread is activated. If the thread has not been allotted a request then the next thread that has been allotted a request will be activated and processed. Figure 2.4 shows the scenario when every thread has been allotted a request.

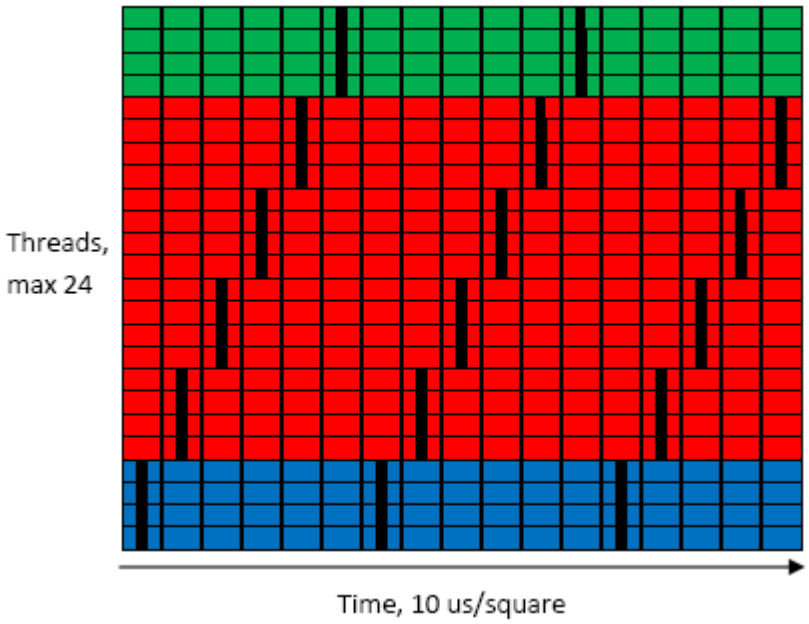


Figure 2.4: Threads

2.8 Capacity

Assuming there is no other power consuming activity, then every request takes approximately 10ms to finish, i.e. every CPU can perform 100 requests every second. In the current hardware there are four CPUs and therefore there is a theoretical maximum of 400 requests per second.
Every session consists of two or sometimes three requests depending on its extension. In average there are approximately 2.2 requests per session, therefore there is a maximum of approximately 182 sessions per second.

2.9 Simulation of the telecom charging system

This chapter is about the simulator used by Ericsson today to run different test cases on the telecom system.

2.9.1 Input

When running simulations a data generator is used as input. The telecom system can handle a maximum of 400 requests per second without being overloaded. The requests are sent in ten groups. This simplifies the process for Ericsson when creating a new test case, since the data generator does not need to be activated as many times as if there were more groups.
This means that if there is a load at 300 requests per second the requests will be sent every 1/10 second and in groups of 30.

2.9.2 Output

The Output handler stores information about the behavior of the queues, the latency times and the number of rejects by the regulator.

2.10 Bottleneck

A bottleneck is the part of a system that works the slowest and therefore may be considered for improvement when modifying the system. For example if there are two subsystems running in sequence and one is running very slowly compared to the other, then it does not matter how fast the faster one operates cause the slow one will not be able to handle all the data to fully use the fast one. The time a request needs to spend in INAPin, TH and INAPout together with the way the threads are distributed causes the bottleneck of the telecom system to be in TH.
The fact that TH is the bottleneck is the reason why the latency regulator is placed before TH to lighten the incoming load to TH.

3 Modeling choices

In this chapter there is a discussion around the choices made when creating the model. The methods used to capture the behavior of the telecom system are described.

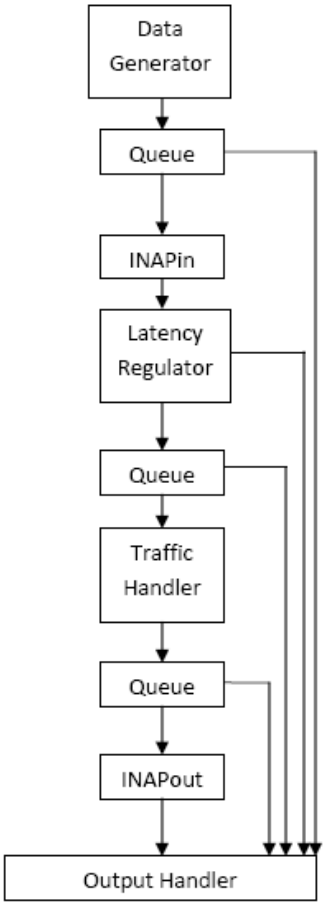


Figure 3.1: Overview

3.1 Modeling approaches

As this thesis is partly about finding a suitable way of modeling a telecom system, I have tried different approaches. Suitable means fast and functional.

3.1.1 Simevents

A first approach tried for model was based on Simevents (see appendix A).

It took very little time to create a first model but there were several limitations. Many built-in functions were used and therefore some of the properties were hidden and not possible to study which I considered to be untenable for future work. Some of the built-in functions had functionality and outputs not used in the model and would probably have caused low simulation speed.

3.1.2 Simulink and RealTime Workshop

The second approach was to create a model from scratch using Simulink with Matlab code (see appendix A). In this way I got a complete control of over all kinds of different properties and behaviors. The coding part was a bit complicated due to the unpleasant Matlab syntax; when coding in Matlab it was a big disadvantage in its handling of vectors and matrixes. These problems caused a bit lengthy solutions, but they worked.
A first validation and testing could at this stage be made which resulted in the expected outcome. As getting closer to a finished model I experienced another problem; the speed. A simulation in Simulink took 40 times the simulated time, i.e. one minute of time simulated took 40 minutes to complete. The model was in need of speed and I investigated the opportunities of optimizing. Several previous cases were found at MathWorks [1]. Most of the methods found involved hand optimization of Matlab code. This optimization reduces the simulation time very little.
The one with the best outcome was using RTW (Real-Time Workshop) which generates optimized c++ code out of the Matlab code. The simulation went much faster, 25% and down to 30 minutes completion time for a one minute simulation.

3.1.3 C++

The third approach was pure c++ coding which finally gave good results. The biggest disadvantage of Matlab compared to c++, I consider being the syntax and its unpleasant abilities in handling variable lengths vectors.
The number of lines of code decreased compare to the previous approach. One minute of simulation took 43 seconds, i.e. 72%. After some code discussion and further optimization with my supervisor I was down to 60% which was considered to be sufficient.

3.2 Limitations

As most complex systems, a charging system includes many smaller subsystems since there are many functions that may differ a lot. A complete model of the charging system would take much time to complete and the model therefore consists of only two subsystems; INAP (Intelligent Network Application Part) and TH (Traffic Handler)
Smaller tasks like for example packing/unpacking are taken in consideration by adding some extra time to the requests.
The subsystems in the model (INAP and TH) do not contain any depth i.e. each request is only considered to spend a specified amount of time in the subsystems of the model and no consideration have been given to the assignments they are performing.

3.2.1 Time of request

Requests entering the system may differ in times spent in the different subsystems. As a simplification there is always a total request time of 10 ms distributed as 2:7:1 between INAPin, TH and INAPout.

3.2.2 No external calls

External calls are not implemented in the model. Therefore the model is independent of other systems. This might cause the load to be more stable than it should be. Statistically there are very few external calls made and therefore it is considered to be unnecessary to include in the model. One

way to include external calls would be to let every 100 000th request or so have an extra long request time in TH, i.e. increase the 7ms that it takes for TH to finish a request.

3.2.3 Priority and Money control

As mentioned in section 2.4 when making a phone call, there is a first balance check to decide whether to accept the call or not. There is also a second request which charges the user. In other words, there are always two requests per session. The second one has a higher priority since it is the one charging the user. When activating the regulator the low priority requests will be rejected before other requests.
This is not the case in the model since there is only one kind of request. This means that there is no difference in prioritization between the requests.
To include prioritization the model needs to remember which of the requests that have been answered. The second request shall be rejected if the first request was rejected, since the user shall not be charged for a call that never went through. To implement this in the model was considered to be too extensive and would not fit within the borders of time of the master thesis.
This limitation might cause the behavior of the model to be more stable than the behavior of the telecom system. Especially when the load is very high since many "charging-requests" will be rejected.

3.2.4 Number of CPUs

When adding a second CPU in the telecom system there is not an increase with 100% in CPU power. The scheduler (section 2.7) is not optimal. There is a scheduling overhead, i.e. it takes some time to handle the CPUs and to distribute the work between them. This has not been taken into account. It is hard to tell exactly how big the scheduling overhead is since no further investigation have been made by Ericsson and it would therefore be hard to implement it in a fair way in the model.

3.3 Data generator

The model needs to know what kind of data to run the simulation with. This decision is made by the user before the start of the simulation. The level of the load, i.e. how high percentage of the maximum 400 requests per second that are used, can be changed every second. A simulation of 1 minute may therefore contain 60 different load levels combined in any desirable way.
The advantages of the data generator in the model are the simplicity and flexibility. It takes much less time to create a new test case and many more scenarios can be tested.
In this chapter the different ways of generating data in the model of the telecom system will be discussed.

3.3.1 100% even method

The easiest way to generate data with a specified load level would probably be to calculate how often a request should enter and at that point create a new request and send it.
If the user wants to run a simulation with a load level at 300 reqs/sec, one way of implementing it would be by sending a new request every 1000ms/300, i.e. approximately every third millisecond. This method gives a result which has little support in real life due to its non existing time variance.

The method was mostly implemented to enable comparison with the two other methods mentioned below. It is also a good idea to use when verifying the behavior of the model since its behavior is rarely caused by the input in this case.

3.3.2 Burst method

As mentioned in section 2.9.1, the data generator used by Ericsson when testing the telecom system, uses an input with bursts. It is of interest to find out how this method affects the output and it is therefore included.

3.3.3 Poisson method

In real life the requests arrive with a variation in frequency in a random way. According to previous theses in similar models [2], a Poisson distribution corresponds well to reality and is therefore used. See appendix B for an example.

3.4 Scheduling and Time slices

Every request that enters the model carries information about how long time it needs to be processed in the different subsystems to finish, i.e. three processing times. If there was no other activity going on, these specified processing times would together be the same as the latency of the request.

INAPin, TH and INAPout are considered as delaying units. The model captures the timing of the telecom system not the logic or function.

A time slice is a short period of time. Every slice is appointed a thread, and the thread will subtract a value equal to the size of the time slice from the processing time of the request that the thread is holding.

With other words, every time a thread is allotted a time slice, the remaining processing time of its request will be subtracted and updated until it has reach zero and leaves for next subsystems that follows according to figure 3.1. When all three processing times have reached zero the Output handler takes care of the request and stores information about its latency.

As mentioned in section 2.7.3 the CPU power is shared between the threads which are run simultaneously. The model achieves the same parallelism as in the real system by using time slices. The distribution of time slices work in the same way as the distribution of the CPU power described in section 2.7.3 and in figure 2.4.

The smaller size of the time slice that are chosen, the closer to the “real thing” it gets and the number of operations will increase as well. Experience of previous cases tells a suitable size of the time slice would be 1/100 of the time to be sliced, which in our case would be 1ms/100=10us.

4 Verification approach

This chapter will explain how the verification of the model against the telecom system has been done. Some basic cases that are used as well as scenarios, that are considered important to be able to be handled well and give outputs as expected, are discussed.

4.1 Basic verification cases

The outputs from the cases mentioned below have been discussed with my supervisor. According to the method used (section 1.4) there have been corrections made until the outputs from the verification were considered satisfactory.

To come nearer the real thing a Poisson distribution (see appendix B) is used when generating the load. The following tests were designed to check the basic functionality of the model.

- One simple request
  By studying the case where one request enters the system without any time sharing or resource sharing with other requests, an overview of the time spent in the different subsystems (INAPin, TH and INAPout) is obtained.
- The scheduling
  When sending a few requests densely after each other, the scheduler needs to distribute the time slices between several requests. By studying the remaining processing times (section 3.4) of each request, one can check that the scheduling works as planned.
- The queues
  With four threads in INAPin there is a maximum of four simultaneous requests in INAPin. By sending five or more requests densely after each other, at least one request will be placed in the first queue, in this way the basic behavior of the queues can be studied.
- Load at 75%
  Even though the Poisson distribution used (section 3.3.3) can cause some temporary overload, a load at 75% of the maximum possible (section 2.8) should not involve any big deviations from the expected output compared to a not overloaded system. This gives an overview of a normal load with expected latencies.
- Load at 100%
  The Poisson distribution (section 3.3.3) will cause temporary overload and the latency regulation will kick in. Few rejects will be made by the regulator since it is based on the average latency value which will reduce extremely long latencies. This gives an overview of the behavior of the regulator.
- Overload
  By sending a load at 120% of the capacity, the behavior of the queues and regulator can be studied. Many requests will be placed in the queues until the regulator has had enough time to adjust its integrator (section 2.6.2) to the overload. This gives information about the speed of the regulator, i.e. how long time it needs to adjust to a certain load.

4.2 Real life situations

The cases below cover some of the real life situations that are important to be able to handle well and give outputs as expected. The outputs from these scenarios have been used as verification and as analysis. The expected latencies are known from experience on the telecom charging system and therefore used as material for verification. The expected behavior of the queues and regulator is unknown and therefore considered as material for analysis. See appendix C for results.

The broken line indicates the maximum load that the telecom system can handle without being overloaded. The solid line indicates the load level.

- Soccer Goal
  During a big soccer game there is a higher load than usual (90%). If anything special happens there will most likely be a shorter overload (110% during one minute).

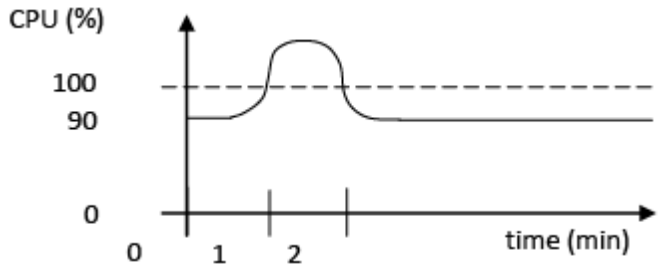


Figure 4.1: Soccer Goal

- New Years Eve
  It is almost impossible to make a phone call during some intensive hours at late New Years Eve. This scenario tests the speed of the latency regulation. There is a much slower increase of the load level compared to the sudden overload in the Soccer Goal scenario.

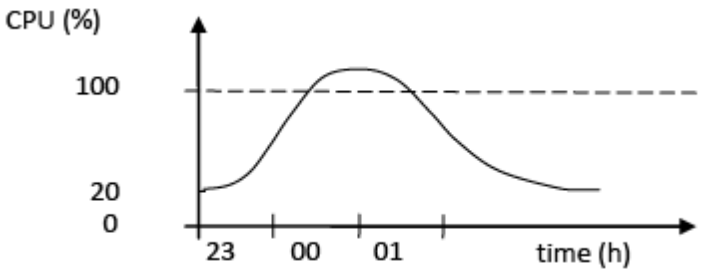


Figure 4.2: New Years Eve

- Hardware Error (Failover)
  Because of hardware error there is no incoming load to the system. This scenario occurs when the CPUs have stopped working for some reason or when doing a restart of the system. When the error is fixed or when the system has completed its restart, there are a lot of irritated customers waiting to make their call and therefore overloads occur in the system.

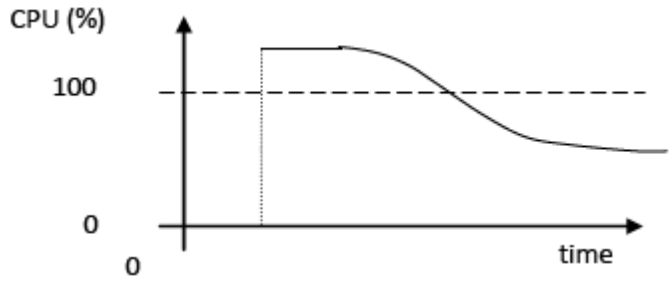


Figure 4.3: Hardware Error

- 99% Load
  This scenario tests how the regulator reacts because of short periods of small temporary overload. The test lasts for 6 minutes but figure 4.4 shows only 5 seconds to give a clear view of how often the load changes

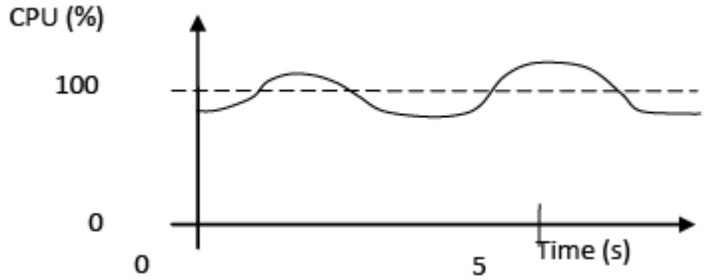


Figure 4.4: 99% Load

- CPU Crash
  This scenario tests what happens when a CPU crashes and there is a temporary shortage of total CPU power (broken line). The load level (solid line) is constant. The CPU power is reduced from 100% to 75% which causes overload.

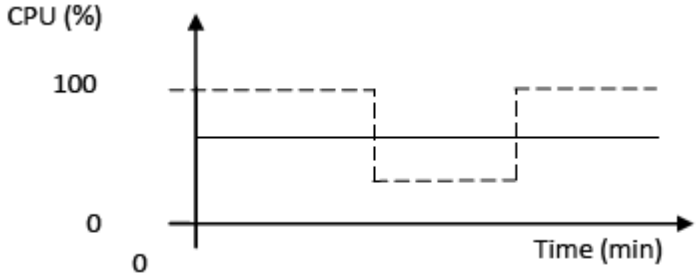


Figure 4.5: CPU Crash

The above tests were generated by writing c++ code that fed the model with the generated traffic. The results can be found in appendix C. The resulting latencies were used to verify the model against the telecom charging system. The information obtained about the queues and the regulator is discussed in the next chapter.

5 Result

The purpose of this chapter is to explain the behavior of every single part of the model by discussing test cases suitable for that single part. Finally the different ways of generating input data and how different software parameters may influence the latencies are discussed.

The Poisson distribution (section 3.3.3) has been excluded in the pictures showing the load level that has been used. This is done to make it easier to see the current load level. Poisson is included in all the other pictures.

5.1 Load at 75%

This section intends to describe the behavior of the latencies when not having any overload.

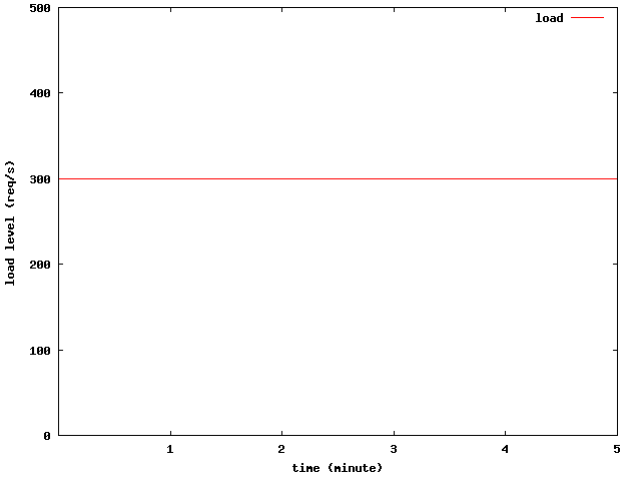


Figure 5.1: Load 75%

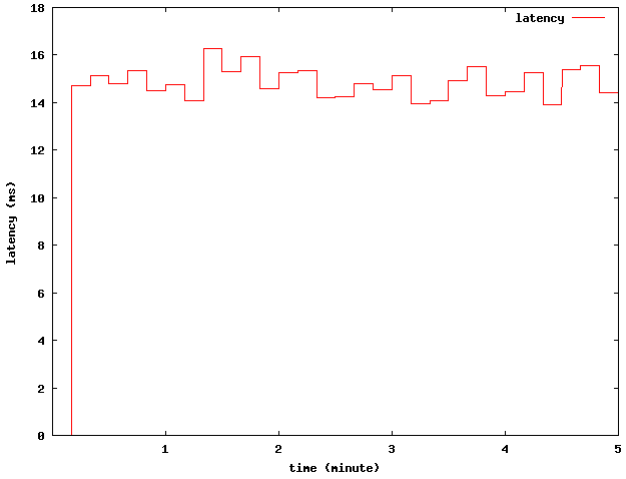


Figure 5.2: Normal latencies (average)

With a load (figure 5.1) at 75% no regulation will take place, since there are low latencies as seen in figure 5.2. The latencies are stable around ~15 ms which is far lower than the lower level used in the latency regulation (section 2.6.2). The variance is caused by the Poisson distribution. Without Poisson there would not be any latency over 10ms. Worth a notice is that the latencies do only contain irregular oscillations, i.e. no clear pattern in the latencies can be detected at a load level of 75%.

5.2 The behavior of the queue

This section intends to describe the behavior of the queue. The bottleneck (section 2.10) of the telecom system is TH, therefore the queue placed before TH is the queue considered to be of most interest. This queue will be referred to as “the queue” in this chapter.

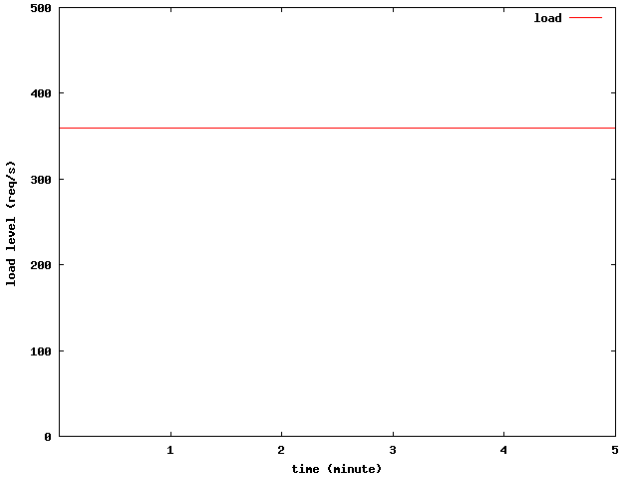


Figure 5.3: Load 90%

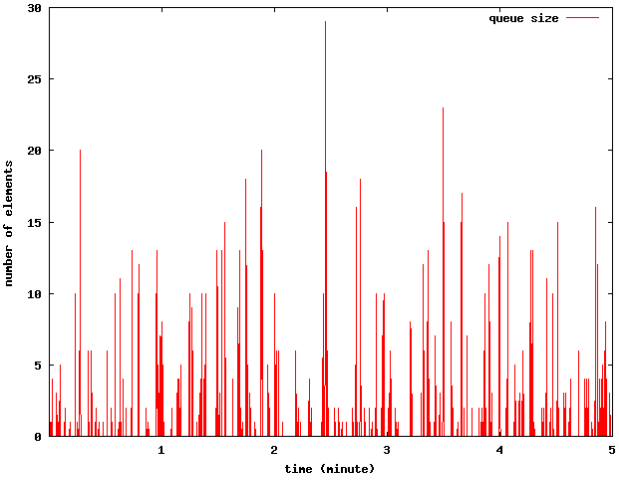


Figure 5.4: Number of requests in the queue

A load (figure 5.3) at approximate 90% will cause the queue before TH (figure 5.4) to temporarily increase, due to the Poisson distribution and its temporary overloading.

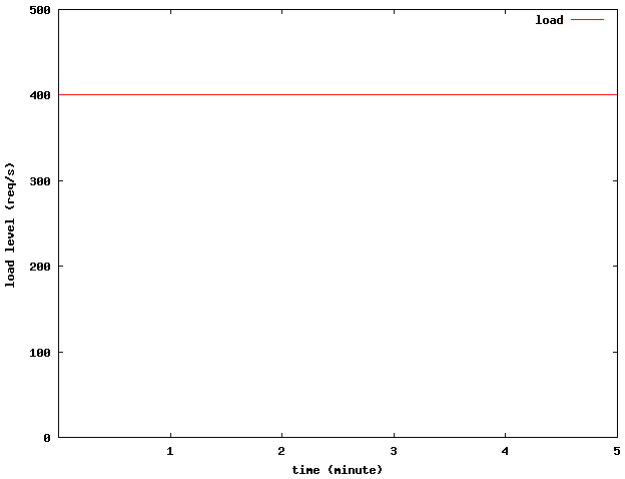


Figure 5.5: Load 100%

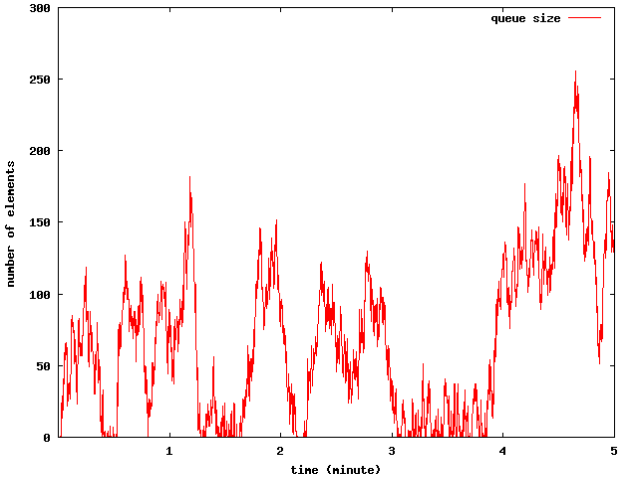


Figure 5.6: Queue

Figure 5.5 shows a load at 100% with no regulation besides the queue method. The data generator delivers an average load level of 100%, i.e. a lower temporary load level will most likely be followed by a higher load level. The time spent waiting for requests to enter during a low load level, will be lost. This is why the queue will have a lot of requests waiting (figure 5.6) when the load goes high.

5.3 Regulation

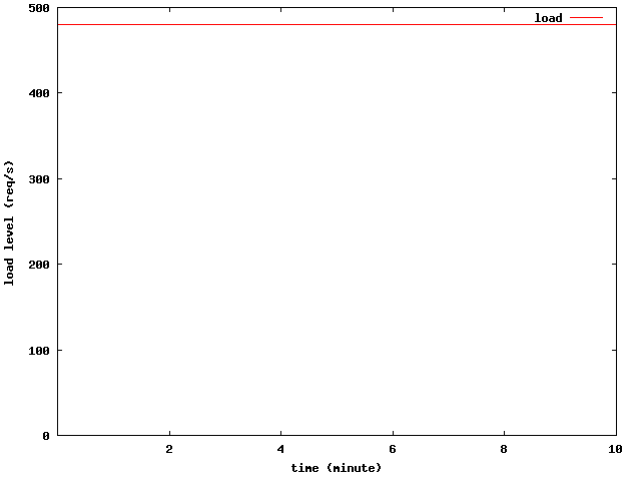


Figure 5.7: Load 120%

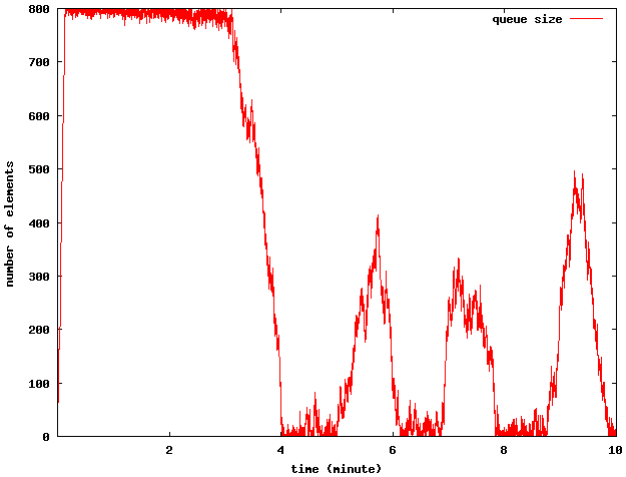


Figure 5.8: Queue

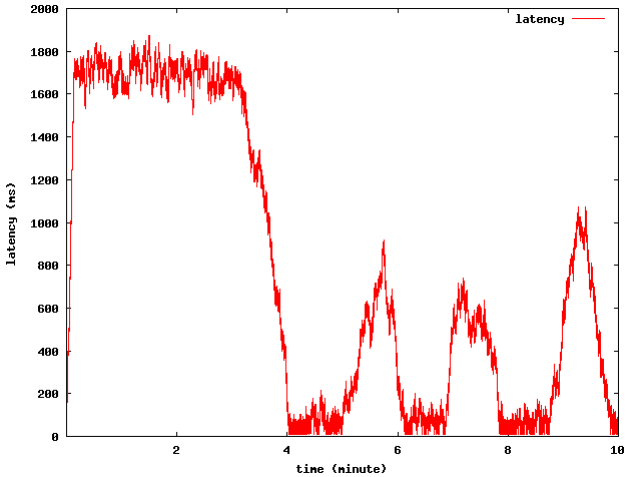


Figure 5.9: Latencies

An important property is the speed of the regulation. One way to test this in the telecom system is by sending a load level at 0% followed by an overload. The load level at 0% is supposed to clear the telecom system from requests and to reset the integrator (section 2.6.2). Since the model is cleared every time the simulation starts, this is not necessary, i.e. there is no need of having the load of 0% in the beginning.

In figure 5.7 the load level of 120% is shown. The big difference in load levels, going from no load to 120%, results in a problem for the regulator. Figure 5.8 shows the number of request in the queue. As seen, it reaches the maximum of 800 which means that the queue method (section 2.6.1) will reject requests.

The overload causes long latencies which activates the latency time method (section 2.6.2). The integrator (figure 5.11) needs some time to adjust to the overload and after a while it finds a suitable level that prevents the queue from being filled (figure 5.8). In this case there is an overload of 20%

and to avoid the latencies from increasing there is a need of rejecting approximately 17% of the calls, since 83% out of 120% is 100%.
As seen in figure 5.11, the integrator level is oscillating around approximately 80% due to the overload at 120%, i.e. 80% of the incoming requests will be accepted and 20% will be rejected by the latency regulator. When the integrator has adjusted to the load level, then the number of requests in the queue decreases, which can be seen in figure 5.8. The queue method deactivates and all request rejections will be done by the latency time method. The relation between the two methods can be seen in figure 5.10.
Due to the oscillating integrator level, the latencies and queue sizes oscillates as well.
Another point worth noticing is the relation between the latency and the size of the queue. Figure 5.8 and figure 5.9 are almost identical since the only way to get latency above the 10ms is to be placed in a queue. The queues placed before INAPin and INAPout hold only a few requests at a time.

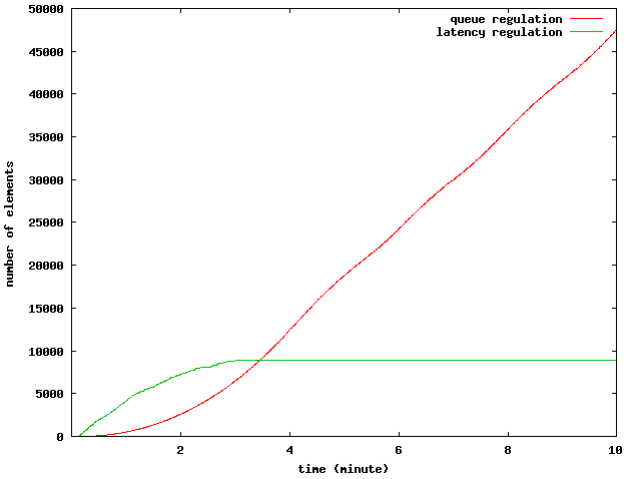


Figure 5.10: Queue and Latency regulation

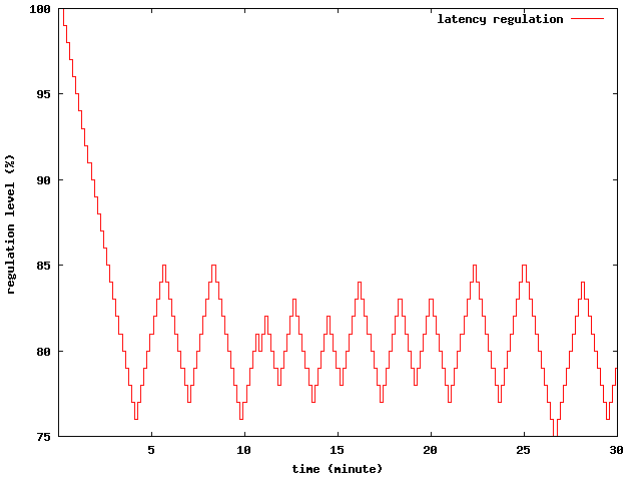


Figure 5.11: Latency regulation

5.4 Input grouping

The data generator used by Ericsson, uses a grouping of ten (section 2.9.1), which means that if there is a load level at 75% of the capacity (section 2.8), i.e. 300 requests per second, every group will consist of 30 requests sent every 0,1 second.
This affects the telecom system by temporally overloading it, and in additional queuing time which results in latencies above the 10 ms (which corresponds to a non overloaded system).
Further tests were performed with grouping to illustrate how much it affects the system. The load is generated with no Poisson distribution. This is done to concentrate on the influence of the grouping.

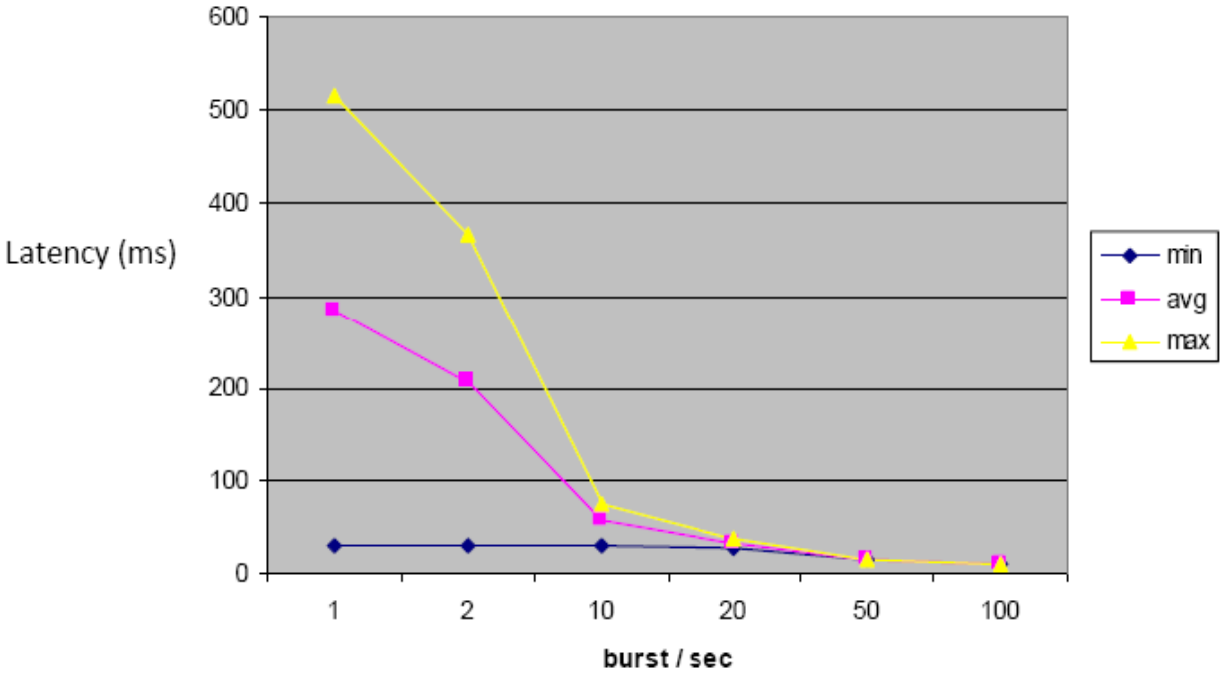


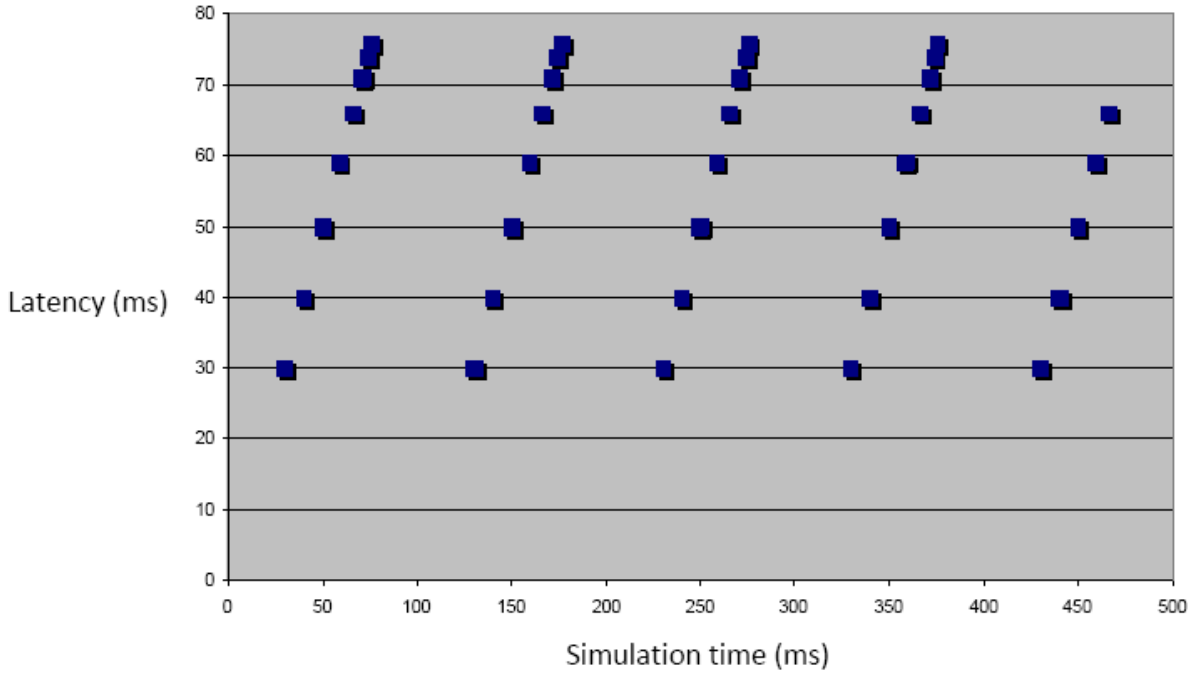
Figure 5.12: Grouping

The case in figure 5.12 has a load level at 300 requests per second which corresponds to 75% and should therefore not cause any overload. The x-axis shows the number of groups, i.e. the number of bursts per second. The more groups the smaller every group gets. The y-axis shows the latency time. With only one burst, all the 300 requests will enter the system at the start of every second. There will still be an average of 300 requests per second but there will be a huge overload for a little while at the start of every second. In this way there will be little or none activity at the end of every second. This is a problem since a load at 75% should not result in any overload, but the latencies will give indications of the opposite.

Number of groups	1	2	10	20	50	100
Avg (ms)	283	206	56	32	14	10
Max (ms)	516	366	75	37	15	10
Min (ms)	29	29	29	26	13	10

Table 5.1: Grouping

There are 4 CPUs in the model, therefore it can process 4 requests simultaneously. This means that if there are 4 or fewer requests per group there will not be any latencies corresponding to overload. A load level at 300 requests per second needs at least 300/4=75 groups to avoid overload. This can be seen in Table 5.1.



Figur 5.13:Example with 10 groups

In figure 5.13 the first 500ms with 75% load and 10 groups can be studied. Every 100th ms a new group of 30 requests enters. Every blue square relates to a request. The x-axis is the simulation time, i.e. it tells when the request is finished. The y-axis is the latency of the request. As seen in Table 5.1 the latencies are between 29ms and 75ms.
Since there are 4 CPUs and all the requests enter the model at the same time, a group of 4 requests will be finished at the same time. As seen in figure 5.13 there are 8 squares during every period of 100ms. The first 7 squares correspond to 4 requests each and the last square corresponds to the remaining 2 requests, i.e. 4*7+2=30.

5.5 Thread distribution

Another property is the number of threads. The threads are distributed as 1:4:1 between INAPin, TH and INAPout. What has been tested is how different multiples of 1:4:1 affects the behavior, i.e. what the difference is between having 6 threads (1:4:1), 12 threads (2:8:2) or 48 threads (8:32:8). The configuration with 24 threads (4:16:4) has very similar outcome to the configuration with 12 threads and is therefore not included in the figure.

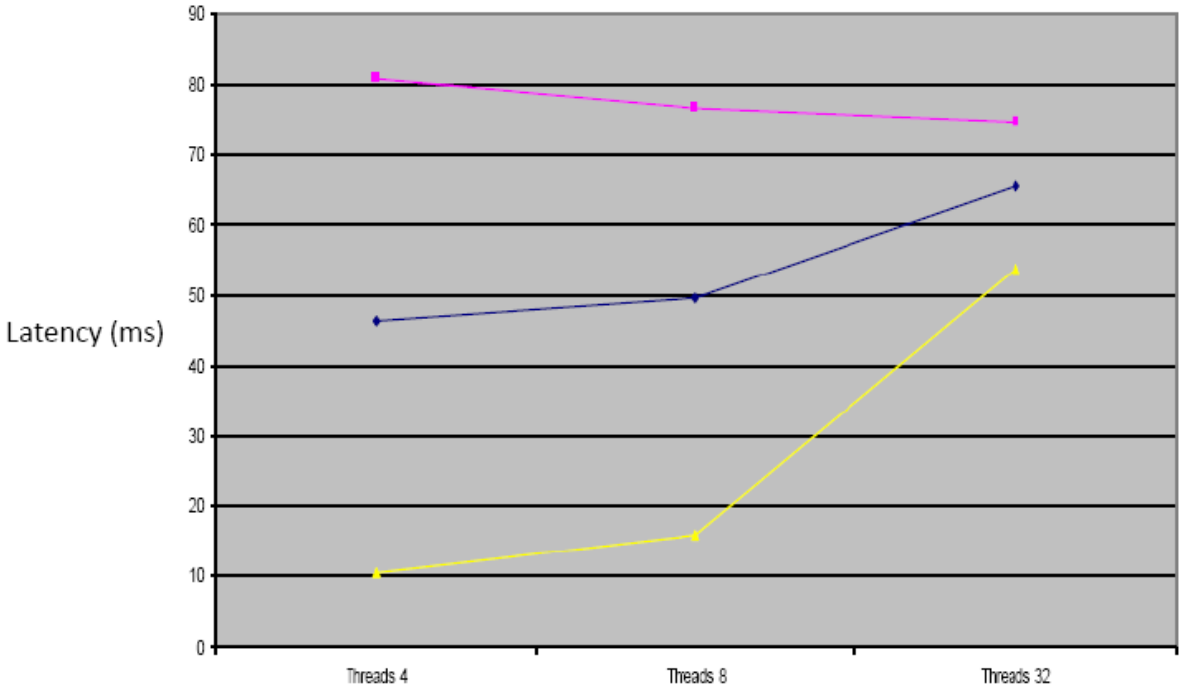


Figure 5.14: Thread distribution

The x-axis in figure 5.14 indicates the number of threads in TH (Traffic Handler). The more threads there is the more requests operates simultaneously, since a requests needs to be allotted a thread to be operated on.
The more threads there are the longer time each thread needs to wait to be allotted a time slice and therefore the minimum latency increases.
The requests are, on the other hand, in an earlier stage allotted a thread, since there are more of them. This eliminates the extremely long latencies and the maximum will decrease.

A load at 75% and a split at 10 give following latencies (ms):

Threads in TH	4	8	32
Avg	46	49	65
Max	80	76	74
Min	10	15	53

Table 5.2: Thread distribution

6 Conclusions

This thesis has illustrated that a model of the telecom charging system can be used to study the load management. What methods are recommended for future similar cases and which should be avoided are discussed below. Also knowledge about the behavior of the models that have been developed are discussed.

6.1 Design

6.1.1 Matlab

The Matlab approach was neglected due to lack of speed. When enabling the ability to simulate different cases and different software specifications, the implementation part became extensive due to lack of support in the Matlab syntax, i.e. the way Matlab code is written.
It was very easy to create a simple model. But when handling different hardware cases, for example different number of CPUs, difficulties occurred. To be able to handle variable length vectors, a lot of extra coding was needed which costs in speed.
The Simevents approach was neglected in an early stage due to its limited view of the behavior. Some of the measure points needed were hidden and the approach was therefore considered unsuitable for further work. Therefore there are few conclusions about the approach.
The RTW (RealTime Workshop) did some successful optimizing but not enough to fix the speed problem.

6.1.2 C++

The code decreased in size when using the c++ approach. My demands and needs were met in a smooth way by the c++ implementation which enabled a high simulation speed and a full view of the behavior.
The vector handling is user friendly which helped a lot while constructing the scheduling process.
The staff at Ericsson are less familiar with Matlab or similar programs and c++ is therefore preferable due to previous experiences.

6.2 Knowledge of behavior

The model enables the ability to study the behavior in an easier way. The simulation time is down at approximate 60% (compared to the existing system at Ericsson) and different kinds of tests (see appendix C) can be simulated without any extensive preparation.

6.2.1 Regulation method

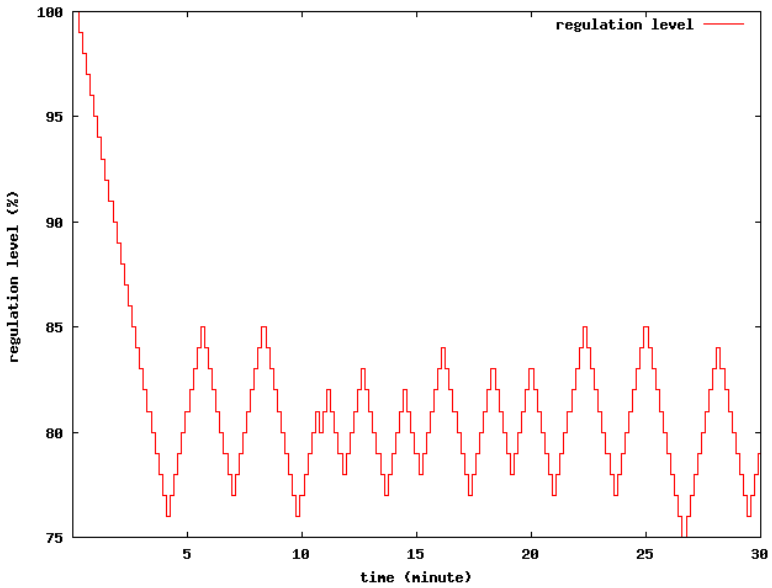


Figure 6.1: Integrator

Figure 6.1 contains the integrator of the latency time method with a load at 120%.
As seen, the integrator is oscillating between 85% and 75% which are considered to be okay. From an optimal point of view it should be stable around approximate 83% since 83% out of 120% is 99,6%. The oscillation causes the number of queued requests to vary a lot. In my test the queue even got filled up and that is why I consider the latency regulation to be in need of reconstruction. The current latency time method is too slow and reacts too late. Some requests are rejected by the queue method which causes longer response time which is considered bad.
The oscillations in the integrator causes oscillations in the latencies and queue size as seen in figure 6.2 and figure 6.3

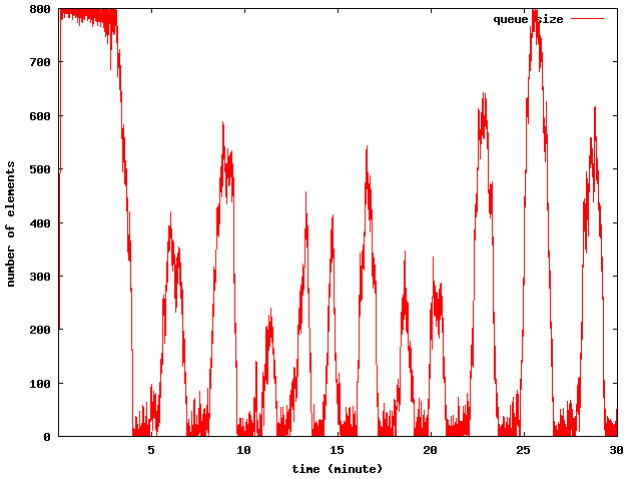


Figure 6.2: Queue before TH

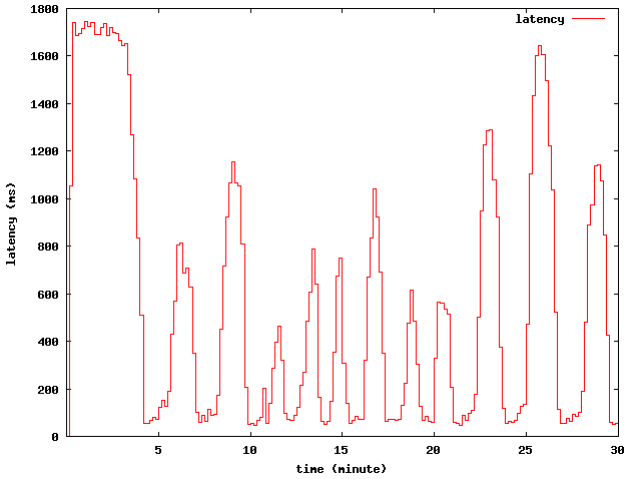


Figure 6.3: Latency

6.3 Analysis

Analysis of the influence of different software and hardware is now possible to be done without extensive or expensive preparations. The simulation only takes 60% compared to the real system.

- Changes in software
  The software can easily be changed. Different thread distributions can be tested (section 5.5).
- Changes in hardware
  The number of CPUs can be changed. Temporary hardware error (section 4.2) can be tested (see appendix C).

6.4 Future work

I consider the latency regulation to be in need of reconstruction. Therefore there is a good opportunity in doing further investigations and discussion around the matter of quality of service, i.e. what defines a good regulation; speed, reliability or perhaps minimization of the response time.
Future work might also include removal of the limitations that I have used; introducing different kinds of requests, adding external calls (see section 3.2).
Further analysis of the behavior might also be interesting. For example optimization of the thread distribution could be done by investigating the distribution of the load between the different subsystems.
More and new measurements can be done. The model enabled the ability to study the behavior of the queues. Similar measurements could be done to get an increased knowledge of the behavior.

7 References

[1] *MathWorks*
http://www.mathworks.com/
2008-10-13

[2] *Uppsatser.se*
http://www.uppsatser.se/om/poisson/
2008-10-13

Appendix A, Glossary

In this appendix there are explanations to some of the word and expressions used in the report.

Mathworks

- Matlab
  One very popular tool when it comes to all kinds of numerical computing tasks is Matlab, which has a wide usage area. It enables plotting of data and functions, implementation of algorithms and more.
- Simulink
  Simulink is a tool for modeling, simulating and analyzing systems. It has a very user friendly interface and offers integration with Matlab. It has several block libraries which are specialized for certain purposes.
- Simevents
  Simevents is a library in Simulink with focus on event simulations i.e. the behavior of the model is based on the activities in it, not the time passed.
- RealTime Workshop (RTW)
  RTW generates C++ code out of algorithms modeled in Simulink or Matlab.

Session & request

A session is a phone call, SMS or any other telecom activity. Every session contains of two different requests, which starts or ends the activity.

Appendix B, Poisson distribution

This appendix includes an example of how the Poisson distribution may influence the latencies.

Example

Below, two cases are illustrated; in the second one the Poisson distribution has been included. The pictures shows the latencies with an overload at 20%; both queue and latency regulation have been included, therefore the slightly decrease in latency.

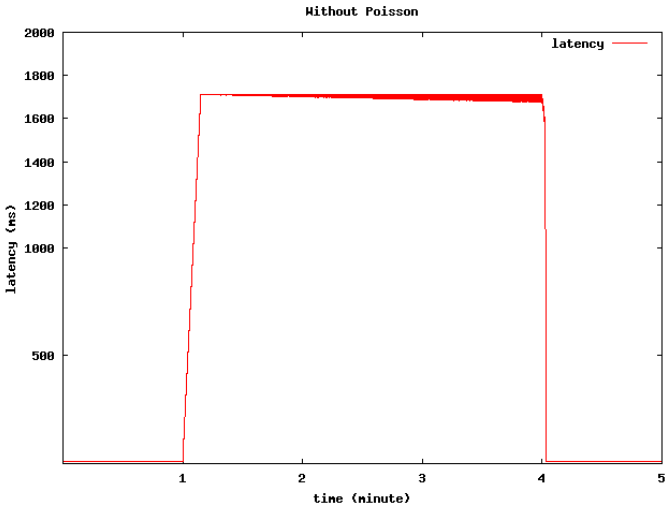


Figure B.1: Poisson excluded

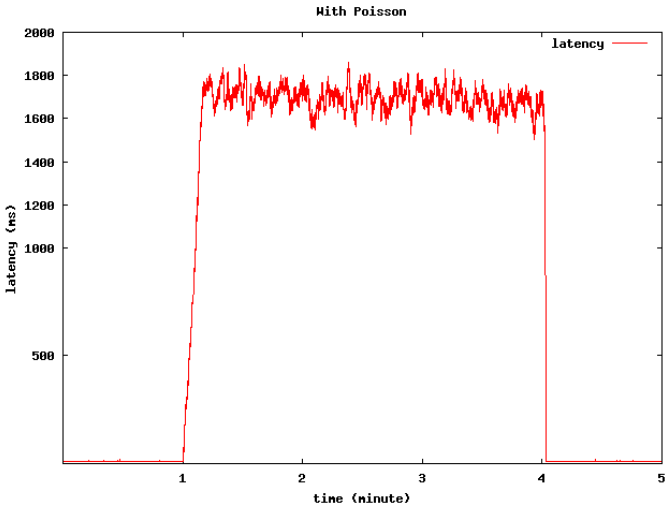


Figure B.2: Poisson included

Appendix C, Simulations

This appendix presents all of the result from the test scenarios in chapter 4.2.

Measurements

There are five different cases studied as mentioned in section 4. Each case is described with six different measurements.

- Load level
  The Poisson distribution has been excluded in the picture to make it easier to see the current load level. Poisson is included in all the other pictures.
- Queue size
  The queue that is observed is the one placed between INAPin and TH, i.e. the one which enables TH to handle temporary overload. The latency regulation is located before TH. TH is also the bottleneck of the telecom system. Therefore this queue is considered to be the only one of interest.
- Latency
  The latencies can be studied both with and without an average method included. The average value is calculated over the last 10 seconds, updated every 10^{th} second.
- Latency time method integrator
  No regulation means 100%, i.e. everything is let through. A heavy load causes a lowered latency integrator level.
- Regulation
  Comparison between the queue method and the latency time method is done. This is done by showing the total number of rejected requests at every point of time.

Soccer Goal

- Load level
  One minute of 120% load

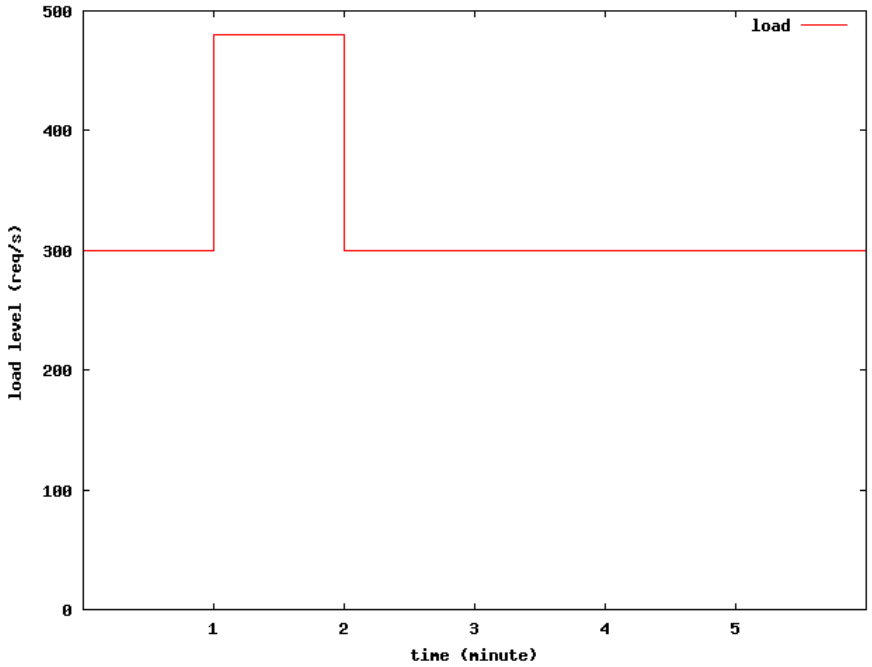


Figure C.1: Load level

- Queue size

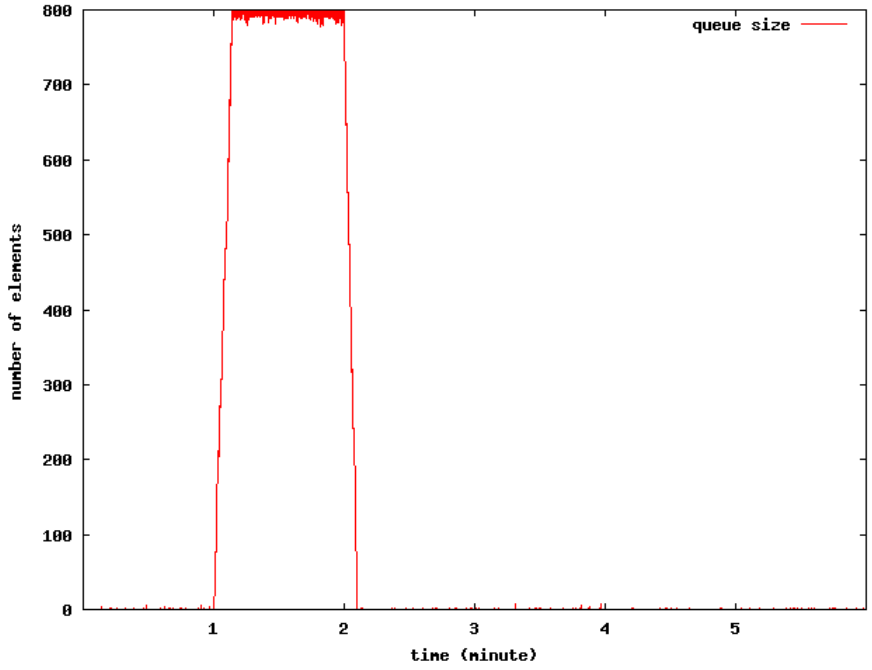


Figure C.2: Queue size

- Latency, no average

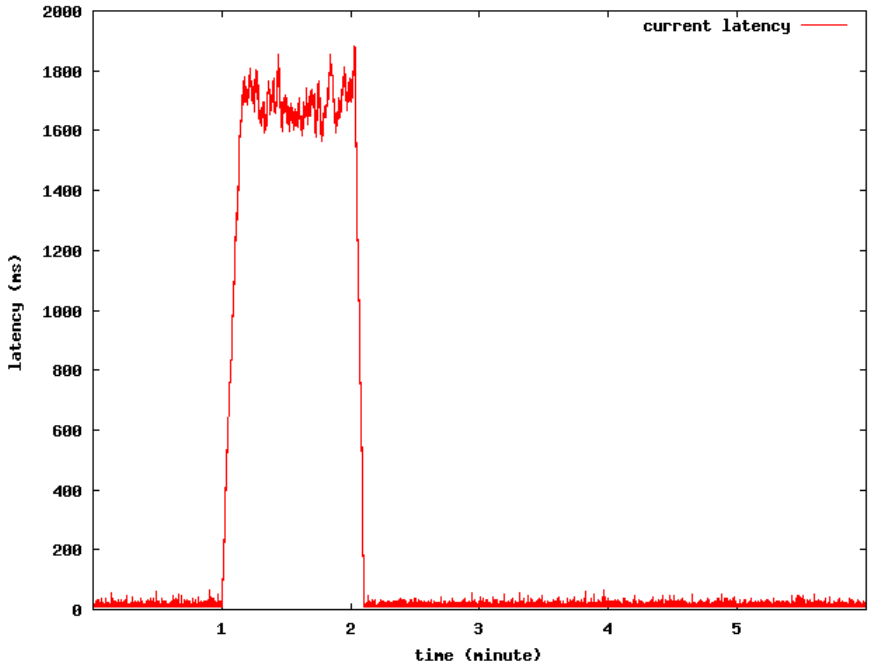


Figure C.3: Latency

- Latency average

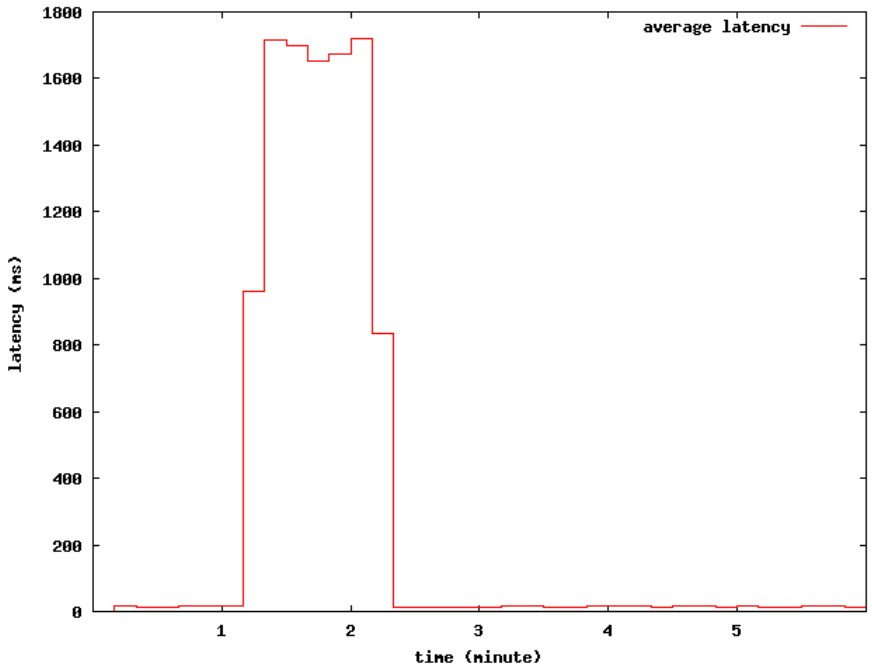


Figure C.4: Latency average

- Latency time method (integrator level)

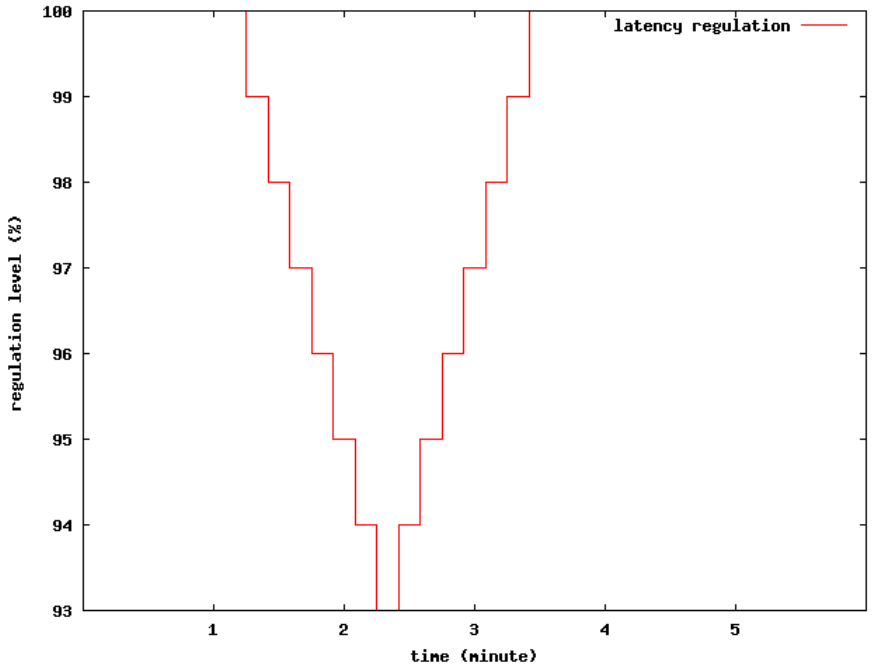


Figure C.5: Latency time method

- Latency time method and queue method

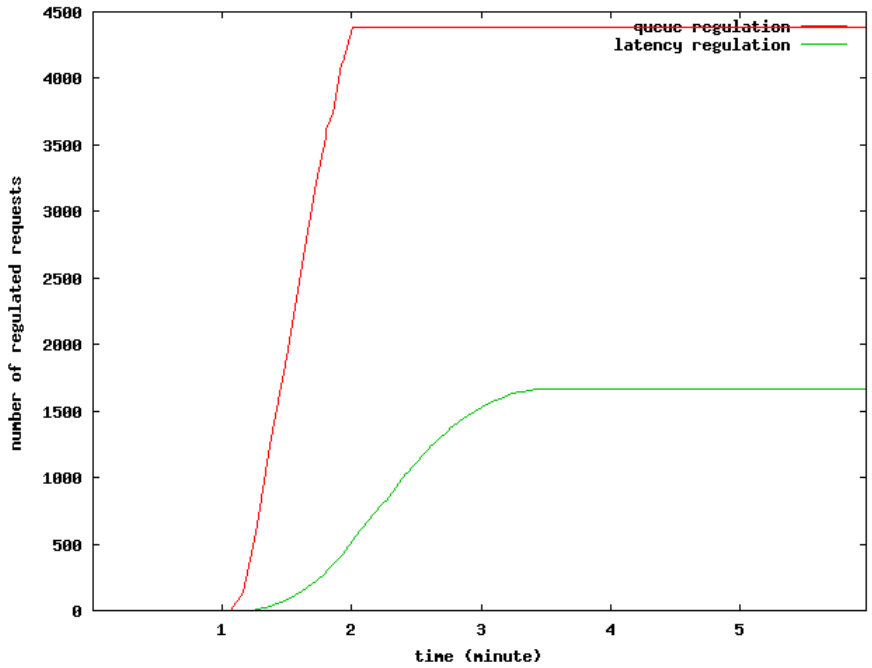


Figure C.6: Regulation

New Years Eve

- Load level

  Slowly increasing load with a maximum at 110%

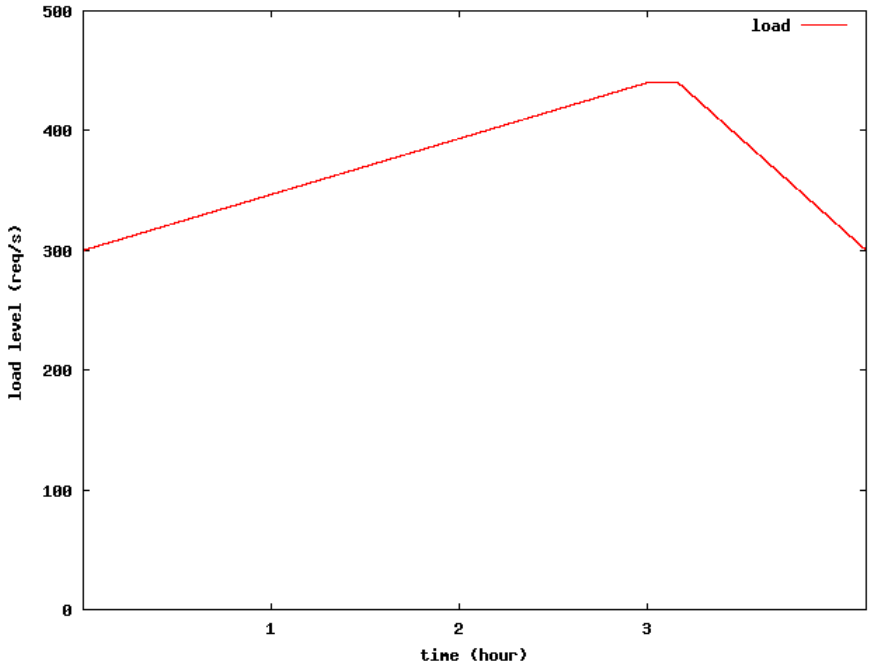


Figure C.7: Load level

- Queue size

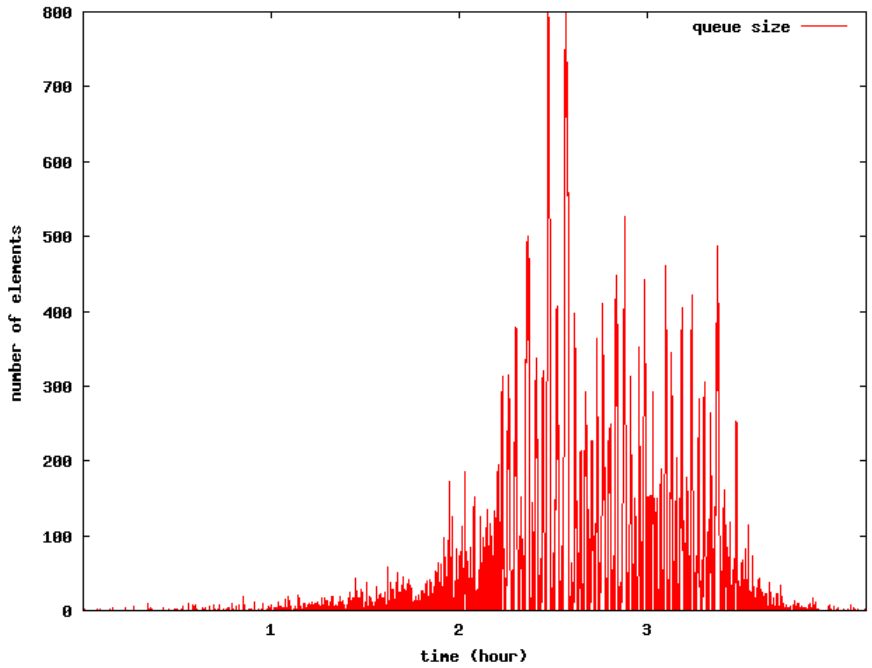


Figure C.8: Queue size

- Latency, no average

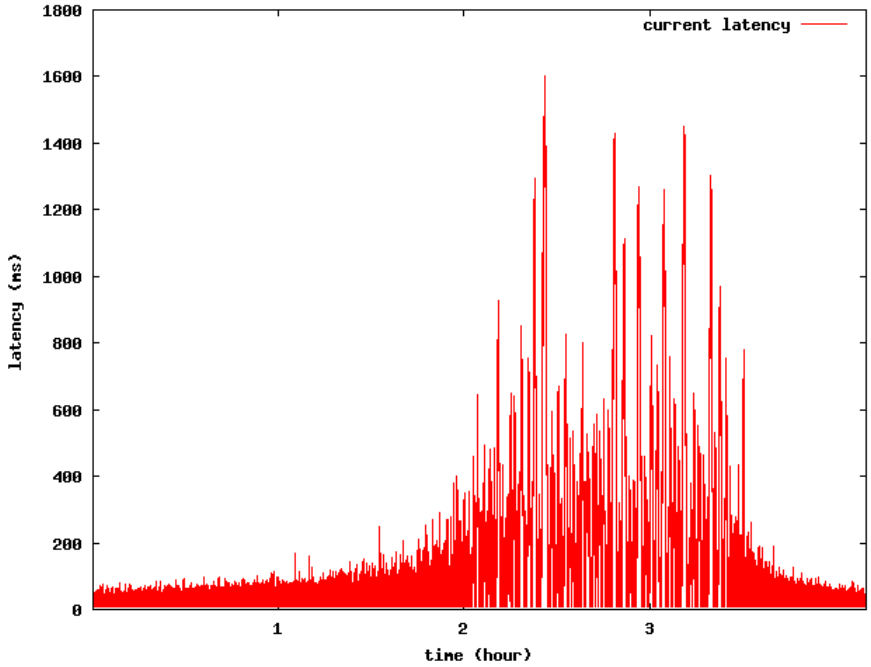


Figure C.9: Latency

- Latency average

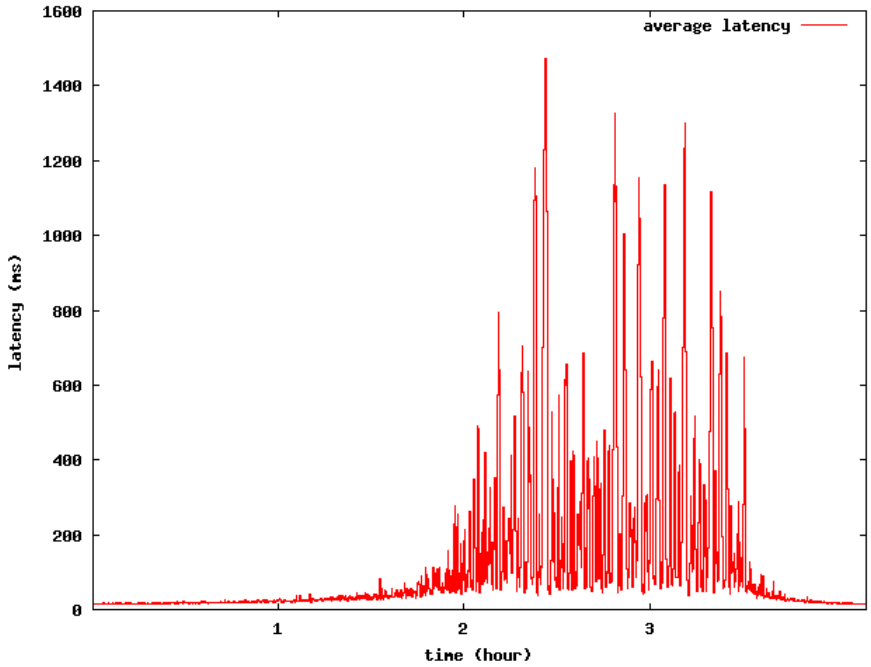


Figure C.10: Latency average

- Latency time method (integrator level)

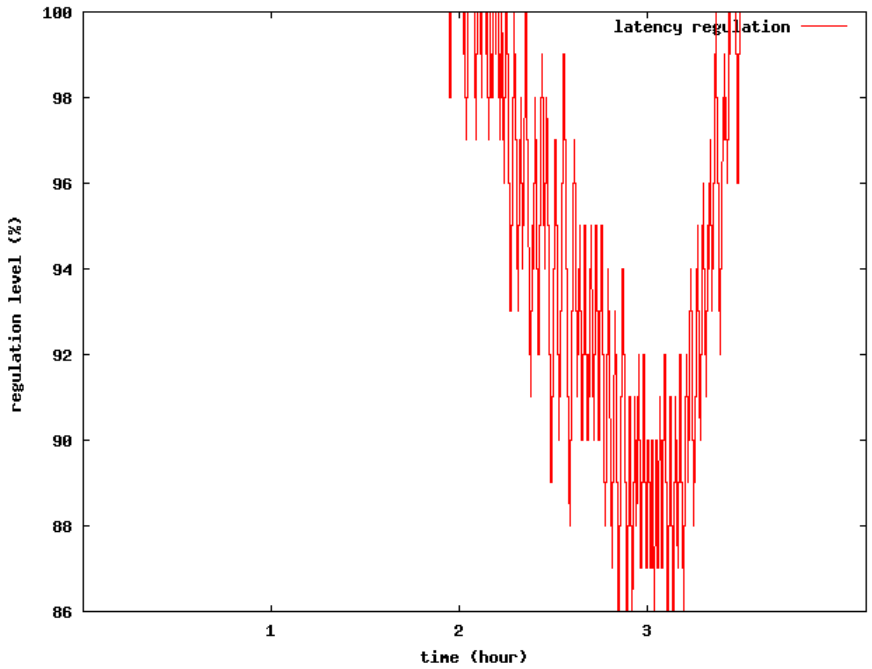


Figure C.11: Latency time method

- Latency time method and queue method

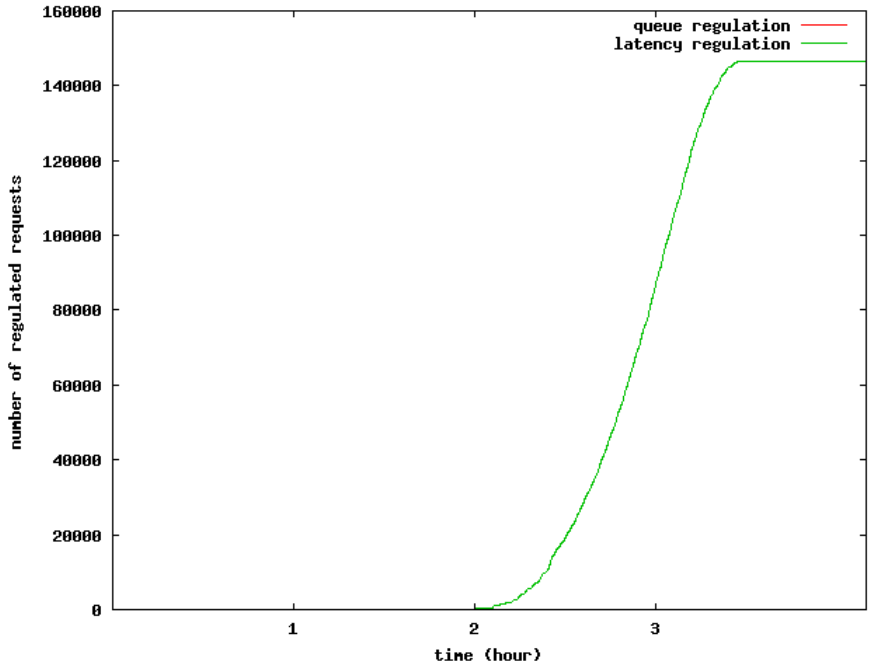


Figure C.12: Regulation

Hardware Error

- Load level

  Load going very quickly from below 1% to 110% and slowly decreasing.

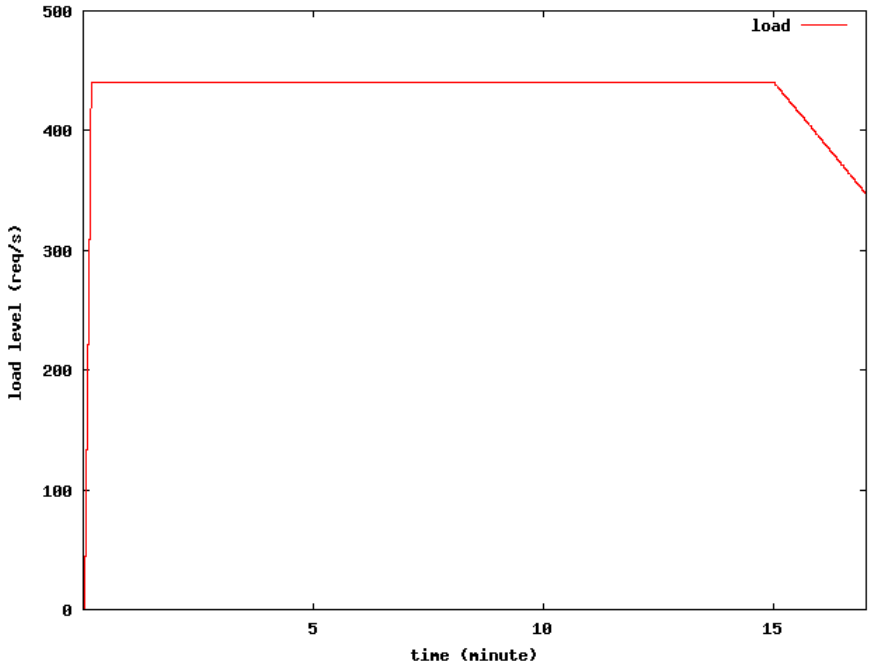


Figure C.13: Load level

- Queue size

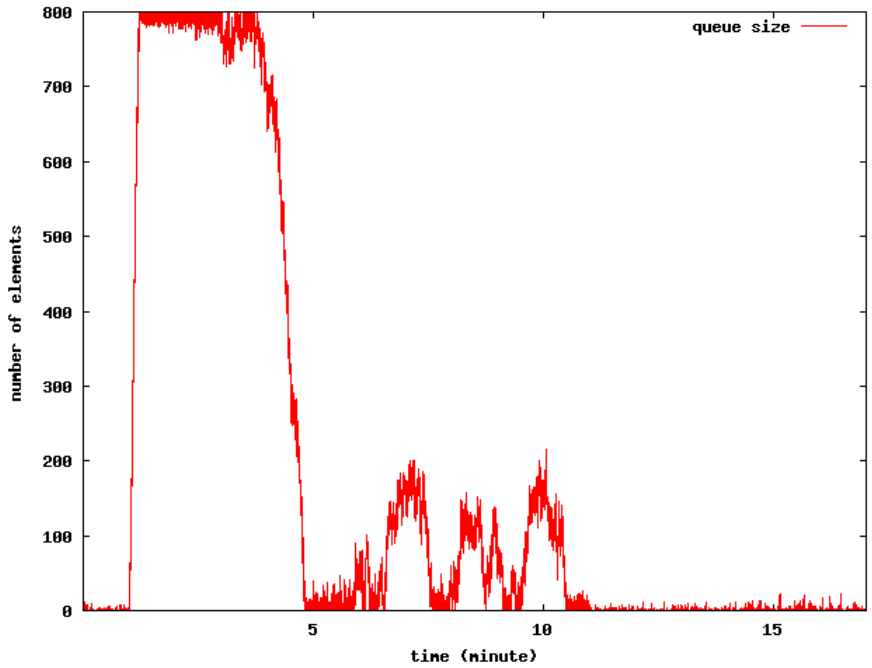


Figure C.14: Queue size

- Latency, no average

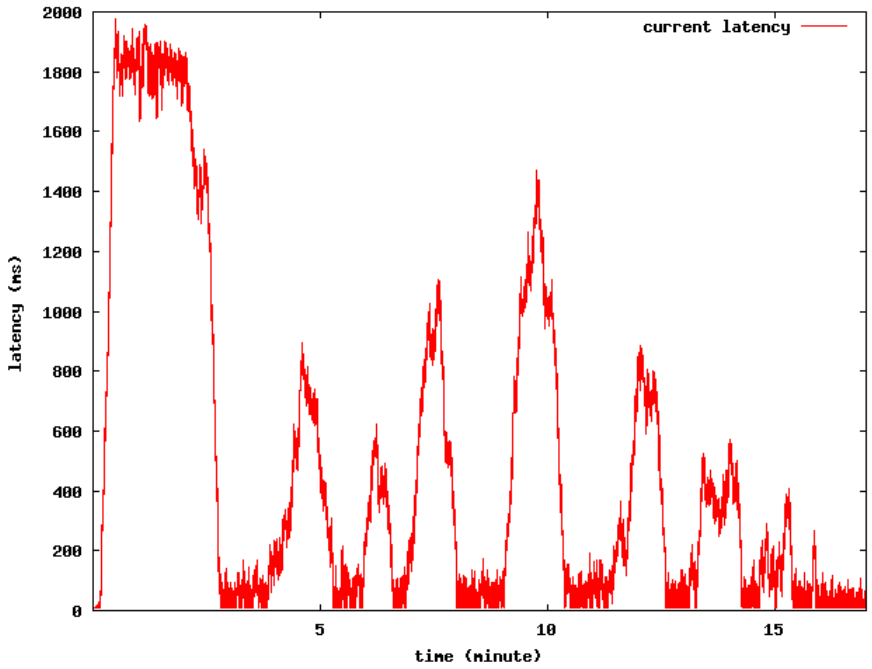


Figure C.15: Latency

- Latency average

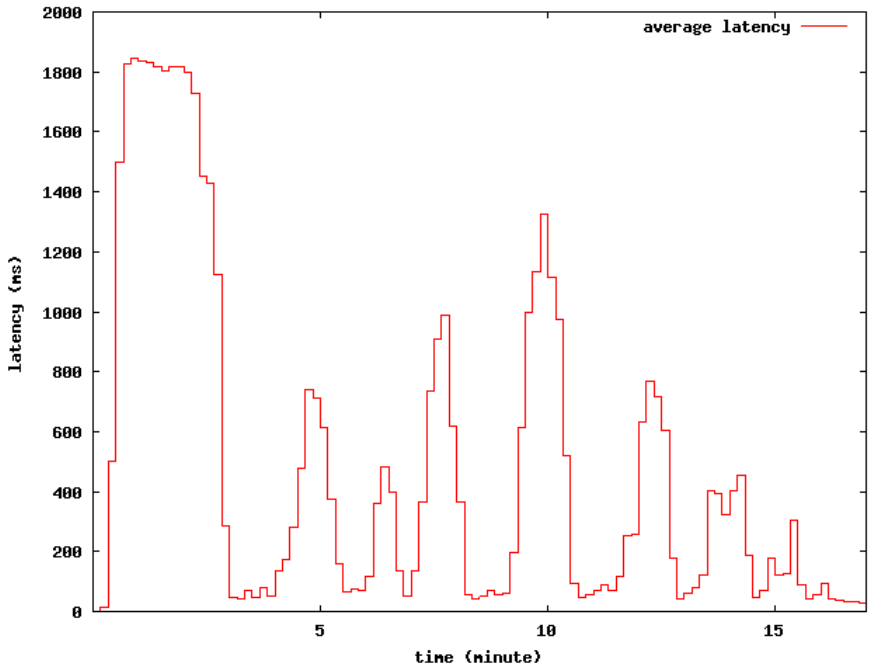


Figure C.16: Latency average

- Latency time method (integrator level)

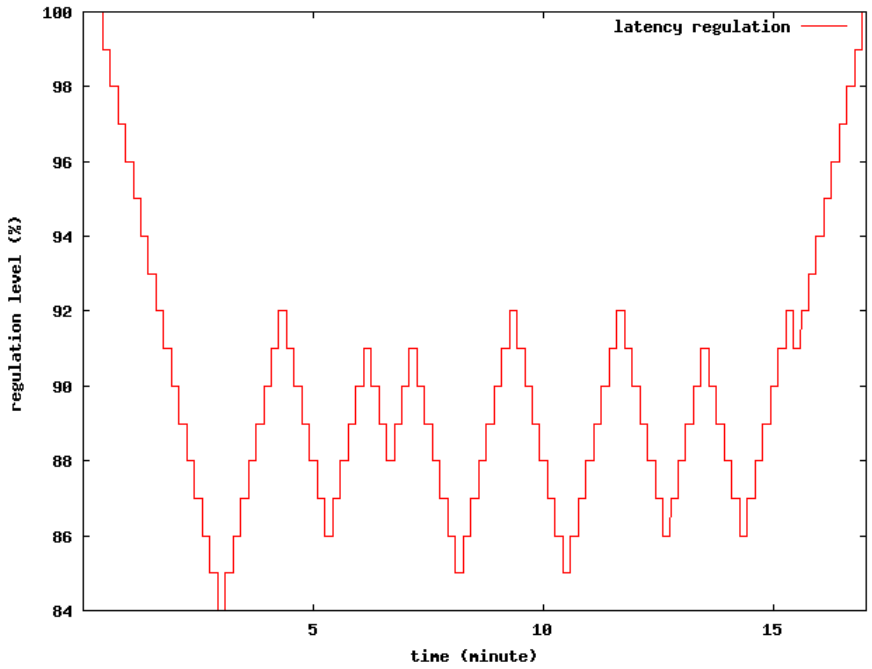


Figure C.17: Latency time method

- Latency time method and queue method

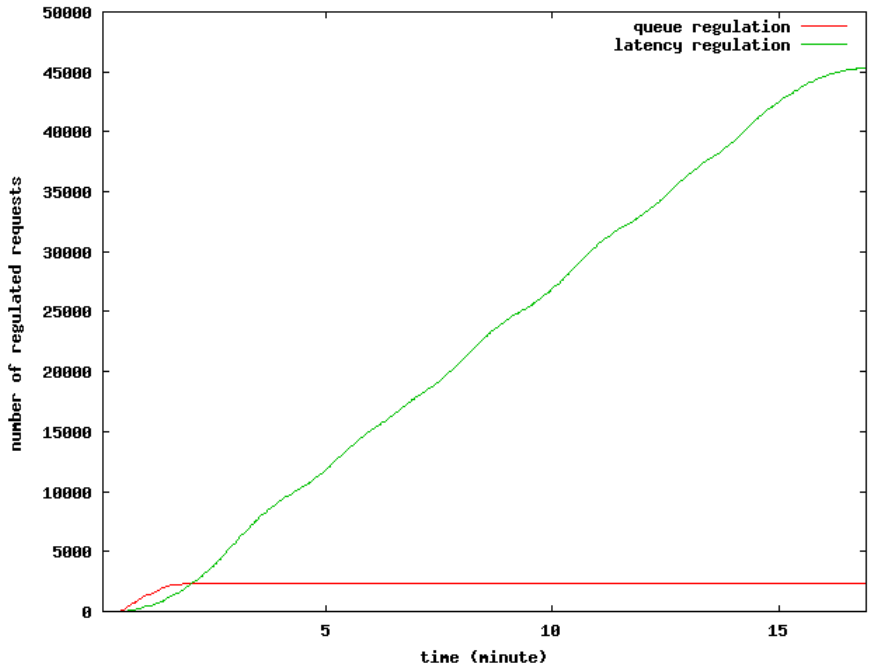


Figure C.18: Regulation

CPU

- Load level

  CPU problem causing a loss of 25% CPU power during one minute.

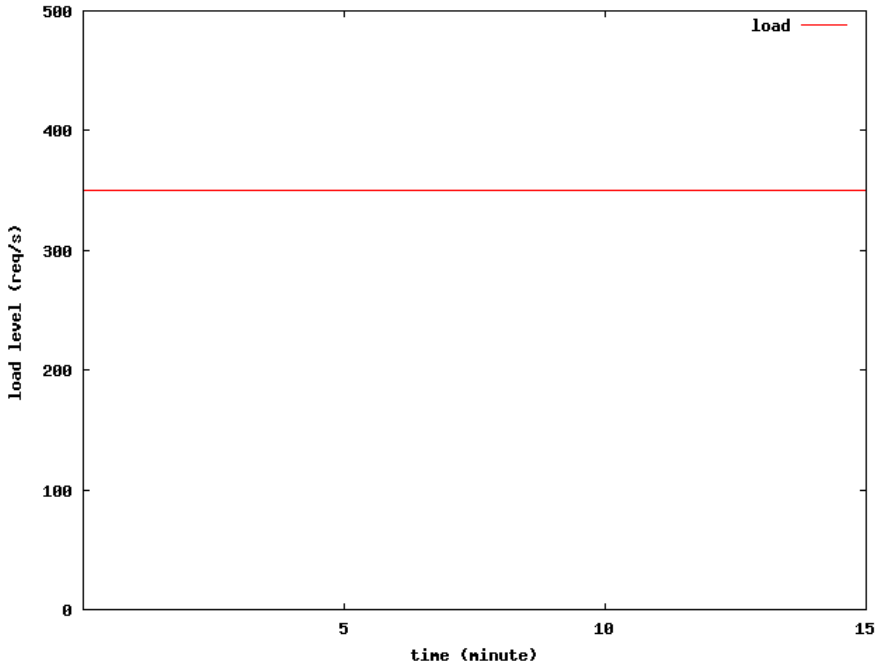


Figure C.19: Load level

- Queue size

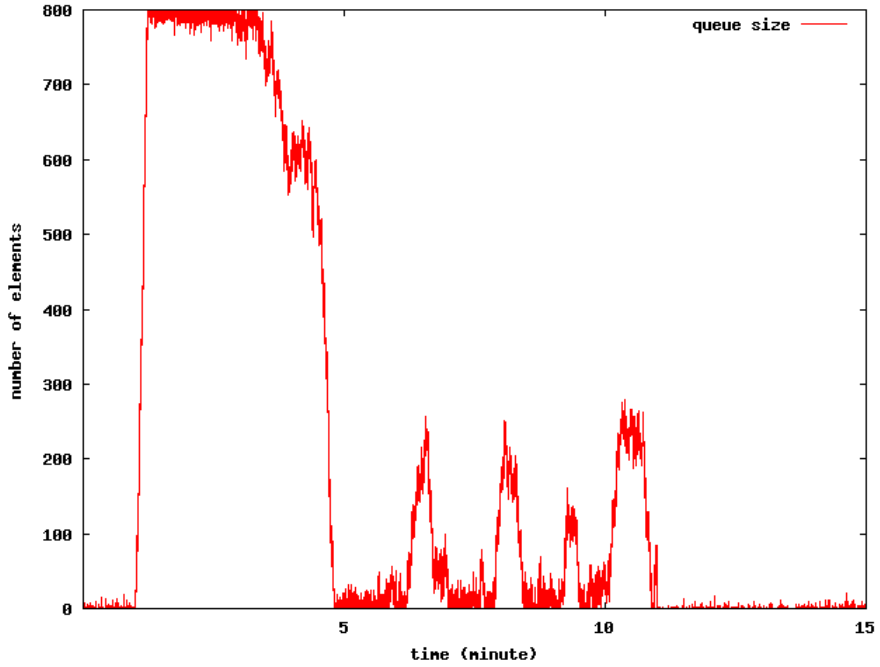


Figure C.20: Queue size

- Latency, no average

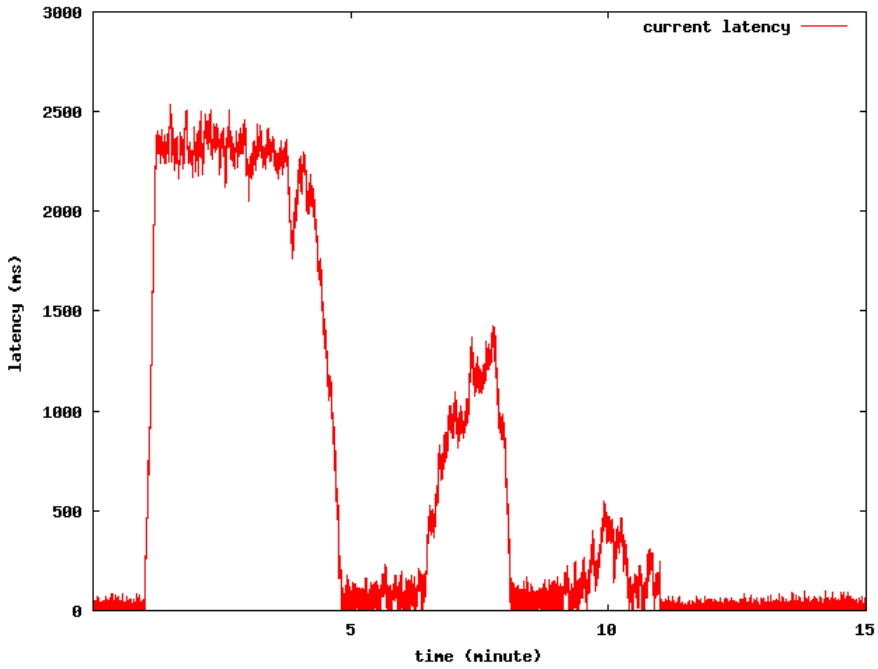


Figure C.21: Latency

- Latency average

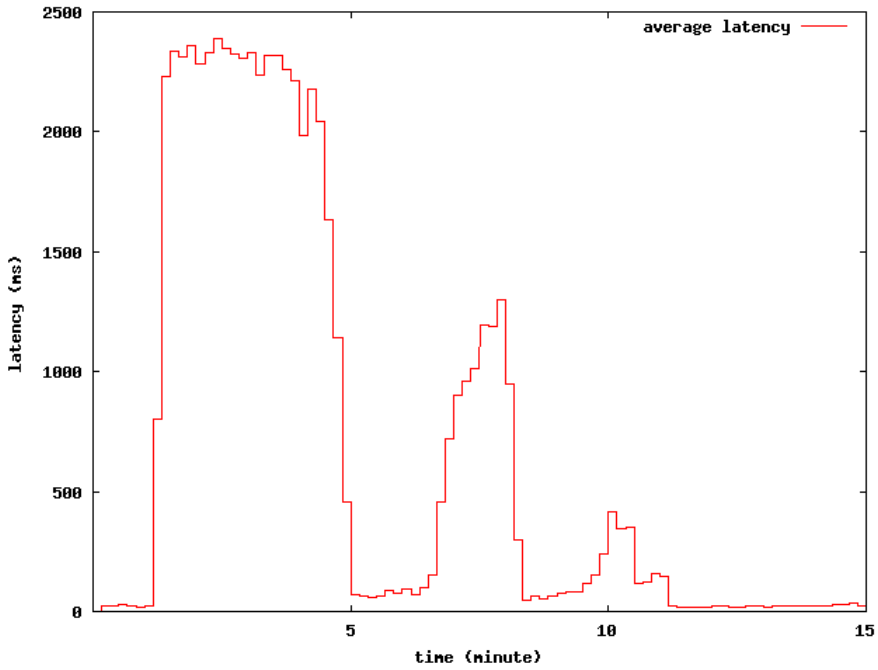


Figure C.22: Latency average

- Latency time method (integrator level)

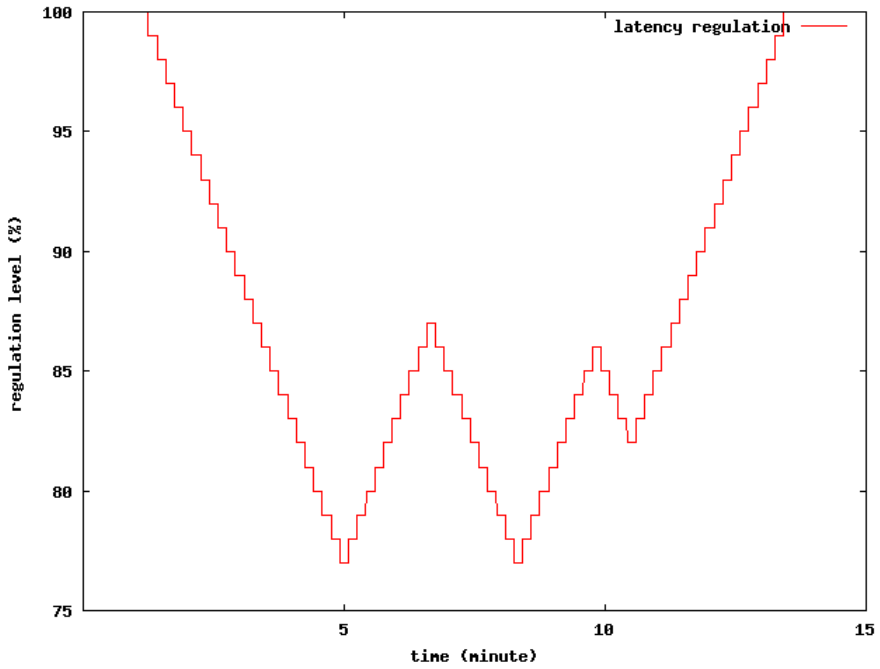


Figure C.23: Latency time method

- Latency time method and queue method

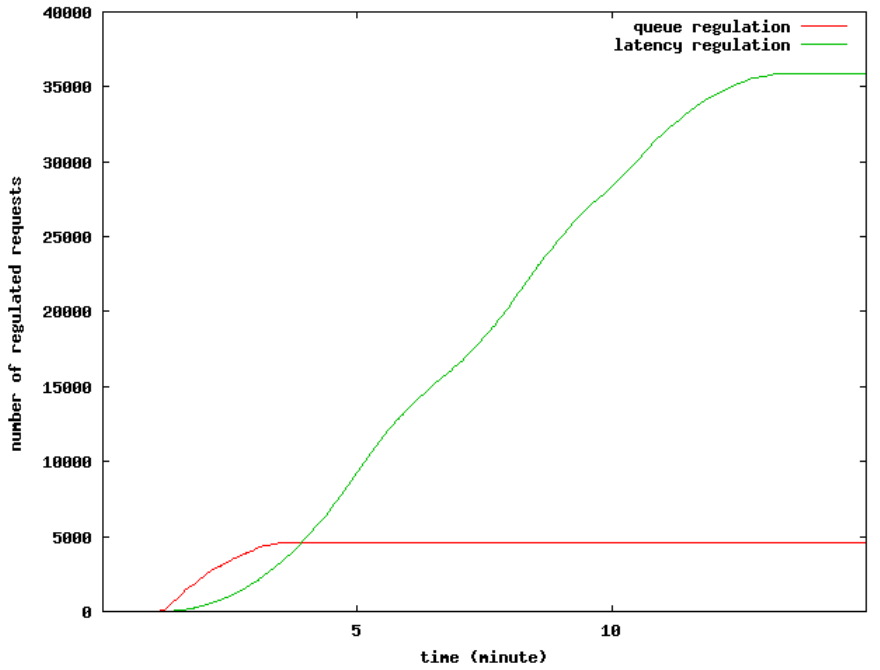


Figure C.24: Regulation

99%

- Load level

  Load at 99% with Poisson causing temporary overload

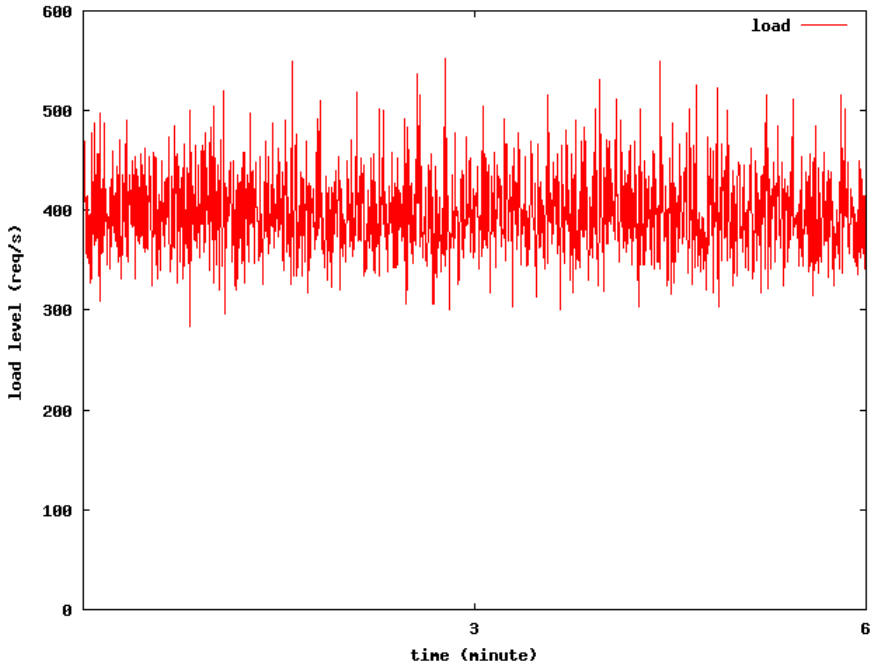


Figure C.25: Load level with Poisson

- Queue size

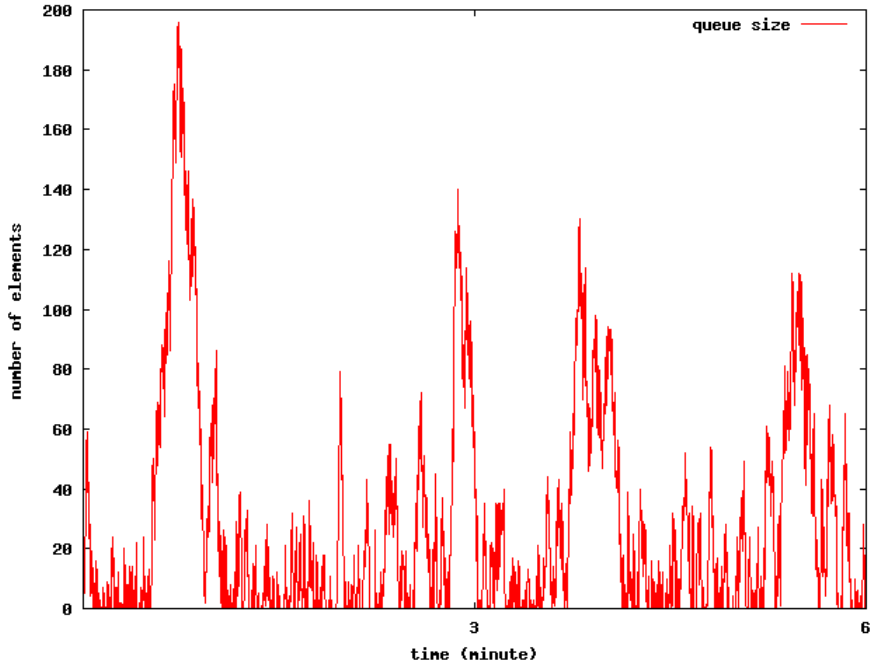


Figure C.26

- Latency, no average

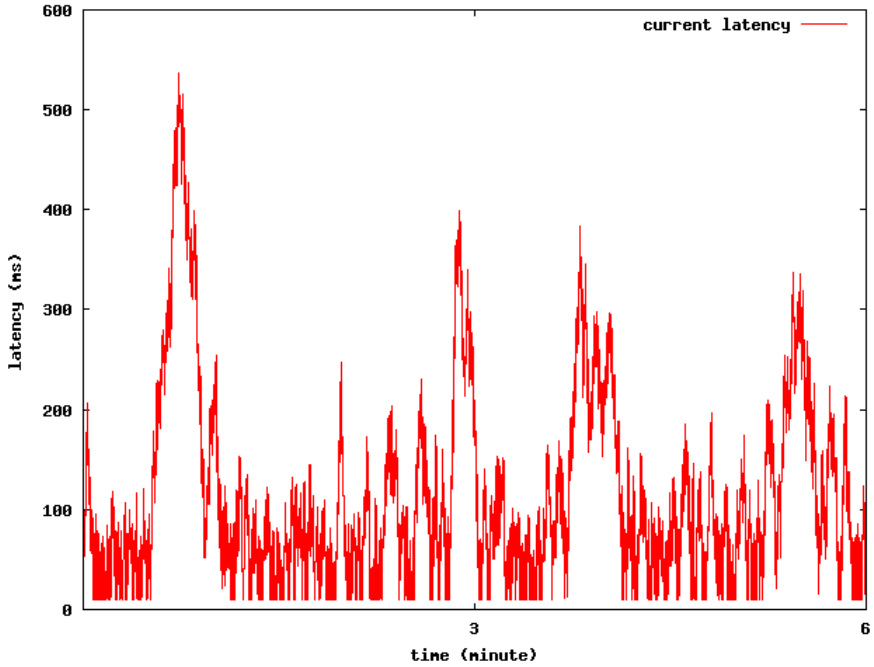


Figure C.27: Latency

- Latency average

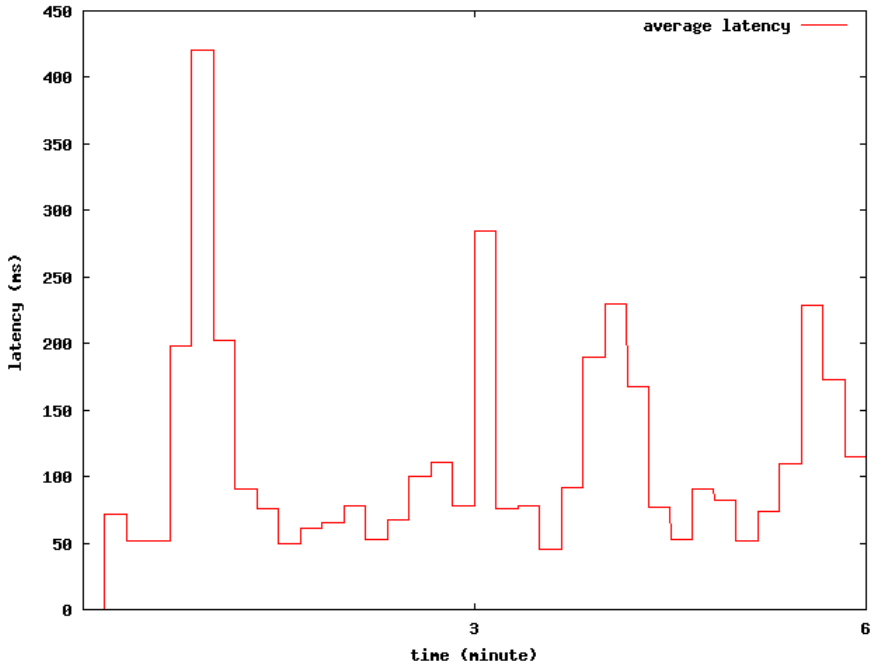


Figure C.28: Latency average

- Latency time method (integrator level)

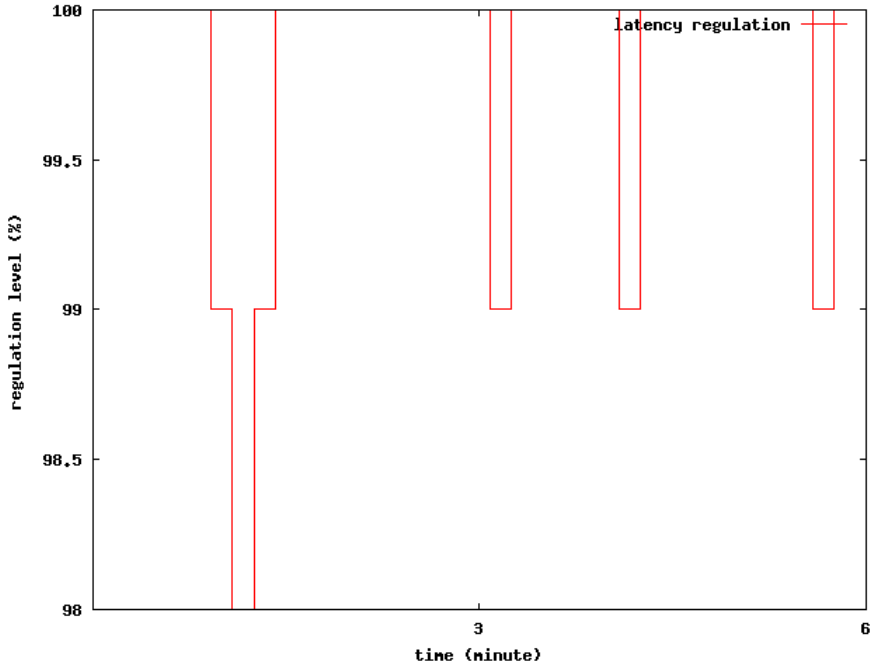


Figure C.29: Latency time method

- Latency time method and queue method

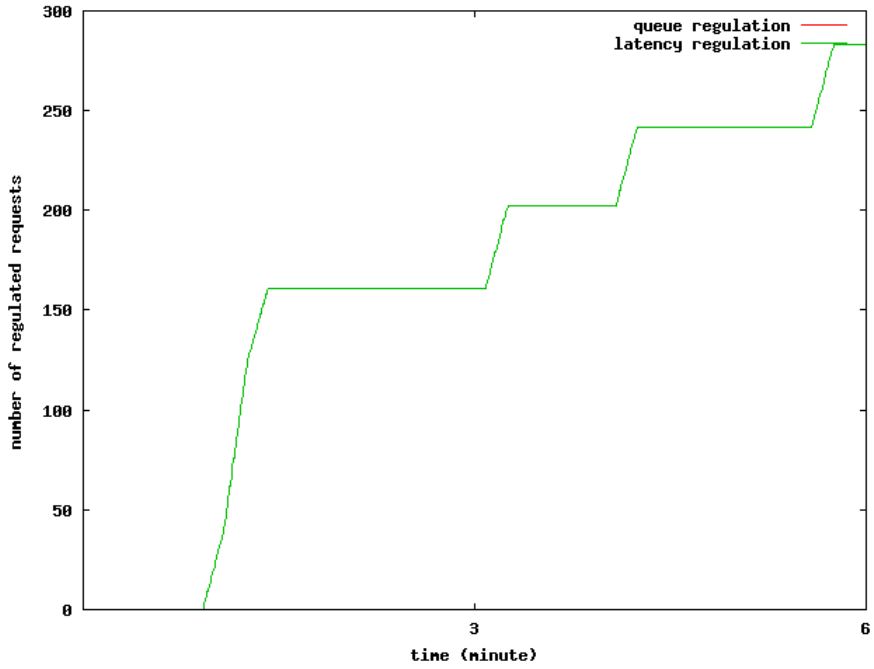


Figure C.30: Regulation

Comments

- Soccer Goal
  The one minute of overload is not enough for the regulator to stabilize.
- New Years Eve
  This is another example of the oscillating latency time integrator. The load is increasing very slowly but the latencies and the queue size are oscillating a lot. As seen in the last picture, the queue method does not act very much which means the latency method is fast enough to be able to handle this case.
- Hardware Error
  Hardware error is causing a stop in all CPUs. When the system goes back on there is a temporary overload which is too high for the regulator to handle.
- CPU
  One CPU crashes, giving 3 CPUs a load of 350 which is about 17% overload. And same thing as in the Soccer goal scenario, the regulator needs to be faster to be able to handle this case in a good way.
- 99%
  The optimal behavior would be not to regulate at all since there is an average load at 99%, which should be of no problem. This is a disadvantage of a latency regulation based on an integrator, since it regulates based on happenings that have come to past.

Appendix D, Specifications

This appendix summarizes the differences between the subsystems from a model point of view.

	INAPin	TH	INA_out
Queue length	200	800	200
Time slice	10us		
Threads	4	16	4
CPUs	4		
Approximate time spent in an unloaded system [ms]	2	7	1

Table D.1: Specifications