Resource monitoring in a Network Embedded Cloud

An extension to OSPF-TE

AMIR ROOZBEH

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Resource monitoring in a Network Embedded Cloud

An extension to OSPF-TE

Amir Roozbeh

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

30 Jun 2013

Examiner: Professor Gerald Q. Maguire Jr.

Abstract

The notions of ***"network embedded cloud"***, also known as a *"network enabled cloud"* or a *"carrier cloud"*, is an emerging technology trend aiming to integrate network services while exploiting the on-demand nature of the cloud paradigm.A network embedded cloud is a distributed cloud environment where data centers are distributed at the edge of the operator's network. Distributing data centers or computing resources across the network introduces topological and geographical locality dependency.

In the case of a network enabled cloud, in addition to the information regarding available processing, memory, and storage capacity, resource management requires information regarding the network's topology and available bandwidth on the links connecting the different nodes of the distributed cloud. This thesis project designed, implemented, and evaluated the use of open shortest path first with traffic engineering (OSPF-TE) for propagating the resource status in a network enabled cloud. The information carried over OSPF-TE are used for network-aware scheduling of virtual machines. In particular, OSPF-TE was extended to convey virtualization and processing related information to all the nodes in the network enabled cloud.

Modeling, emulation, and analysis shows the proposed solution can provide the required data to a cloud management system by sending a data center's resources information in the form of new opaque link-state advertisement with a minimum interval of 5 seconds. In this case, each embedded data centers injects a maximum 38.4 bytes per second of additional traffic in to the network.

Keyword = *network enabled cloud, network embedded cloud, carrier cloud, Cloud LSA, data center model, opaque LSA, link-state update policy, OSPF-TE, resource monitoring, network-aware, cloud management system, extended TED.*

Sammanfattning

Ett network embedded cloud", även känt som ett network enabled cloud"eller ett "carrier cloud", är en ny teknik trend som syftar till att tillhandahålla nätverkstjänster medan on-demand egenskapen av moln-paradigmet utnyttjas. Traditionella telekommunikationsapplikationer bygger ofta på en distributed service model och kan använda ett network enabled cloudsom dess exekverande plattform. Dock kommer sådana inbäddade servrar av naturliga skäl vara geografiskt utspridda, varför de är beroende av topologisk och geografisk lokalisering. Detta ändrar på resurshanteringsproblemet jämfört med resurshantering i datacentrum.

I de fall med ett network enabled cloud, utöver informationen om tillgängliga CPU, RAM och lagring, behöver resursfördelningsfunktionen information om nätverkets topologi och tillgänglig bandbredd på länkarna som förbinder de olika noderna i det distribuerade molnet. Detta examensarbete har utformat, tillämpat och utvärderat ett experiment-orienterad undersökning av användningen av open shortest path first med traffich engineering (OSPF-TE) för resurshantering i det network enabled cloud. I synnerhet utvidgades OSPF-TE till att förmedla virtualisering och behandla relaterad information till alla noder i nätverket.

Detta examensarbete utvärderar genomförbarheten och lämpligheten av denna metod, dess flexibilitet och prestanda. Analysen visade att den föreslagna lösningen kan förse nödvändiga uppgifter till cloud management system genom att skicka ett datacenters resursinformation i form av ny opaque LSA (kallat Cloud LSA) med ett minimumintervall av 5 sekunder och maximal nätverksbelastning av 38,4 byte per sekund per inbäddade data center.

Nyckelord = *network enabled cloud, network embedded cloud, carrier cloud, Cloud LSA, data center modell, opaque LSA, link-state uppdatering policy, OSPF-TE, resurs övervakning, cloud förvaltning systemet, förlängd TED.*

Acknowledgements

First of all, I would like to express my sincere gratitude to my academic supervisor and examiner Professor Gerald Q. Maguire Jr., for all the support, feedback, and encouragement.

I wish to thank also my Ericsson's industrial supervisor Azimeh Sefidcon, and a special gratitude I give to Ericsson TPA research group, Per Karlsson, Srinivasa Vinay Yadhav, Hareesh Puthalath, Enrique Fernandez Casado, and Bob Melander for helping, trusting, and giving me this great opportunity.

Furthermore, I would also like to acknowledge with much appreciation the crucial role of my loving wife Paria Abedi and my sweet friend Jesika, for their unconditional love, supports, and encouragements.

Last, but not least, I would like to express my thankfulness to my parents Mehri Khaleghi and Alireza Roozbeh and my sister Mehrnaz Roozbeh who stood in staunch behind me, and I have been indebted to them for making my dreams come true.

Finally, I would like to share my favorite quotes:

"Nothing Is Impossible; The Word Itself says I'm Possible - Audrey Hepburn."

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ABR	Area Border Router
Ack	Acknowledgment
API	Application Programming Interface
AS	Autonomous System
Bps	Byte per second
BGP	Border Gateway Protocol
CAPEX	Capital Expenses
CI	Confidence Interval
CMS	Cloud Management System
DB	Database
DV	Distance Vector
GB	Gigabyte
GHz	Gigahertz
IGP	Interior Gateway protocol
IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
IaaS	Infrastructure as a Service
IANA	Internet Assigned Numbers Authority

L1VPN	Layer 1 Virtual Private Network
LS	Link State
LSA	Link-State Advertisement
LSDB	Link-State Database
LSU	Link-State Update
MB	Megabytes
MBps	Megabytes per second
MOSPF	Multicast Open Shortest Path First
MPLS	Multiprotocol Label Switching
NIST	National Institute for Standard Technology
NSSA	Not-so-stubby-area
NaaS	Network as a Service
OPEX	Operating Expenses
OSPF	Open Shortest Path First
OSPF-TE	Open Shortest Path First Traffic Engineering
OSR	Open Source Routing
OTT	Over The Top
PaaS	Platform as a Service
PaaS	Process Node
QoE	Quality of Experience
QoS	Quality of Service
RIP	Routing Information Protocol
SaaS	Software as a Service
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol

SPF	Shortest Path First
TCP	Transmission Control Protocol
TE	Traffic Engineering
TE-LSA	Traffic Engineering Link-State Advertisement
TED	Traffic Engineering database
TLV	Type/Length/Value
VM	Virtual Machine
WAN	Wide Area Network
XaaS	Everything as a Service

Chapter 1

Introduction

This chapter provides an introduction to the subject of this thesis project in order to help readers understand the scope of this study. This chapter is organized as follows. The first section gives a summary of cloud computing and cloud networking and provides a foundation for understanding the current challenges of these areas. Next, the problems addressed in this thesis project are described, followed by a proposed solution and a statement of the project's goals. Moreover, the limitations of this work is discussed briefly. The chapter concludes with an outline of the thesis.

1.1 Overview

cloud technology is a hot and rapidly evolving topic. It provides scalable virtual or physical resources over the network according to one of several different service models. More specifically, cloud computing can be classified according to these service models into: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), Network as a Service(NaaS), or more generally "Everything as a service" (XaaS) [1]. Cloud characteristics include on-demand services, broad-band network access, and pay-as-you-go charging for the services used. These three characteristics enable companies to reduce both their capital and operating expenses (CAPEX and OPEX) [2]. In addition, companies can access almost unlimited resources on demand without needing to consider the maintenance, security, and other aspect of the underlying computational infrastructure by using cloud services.

From a topology perspective, a cloud can be classified into one of three types [3]: ***Centralized cloud***, ***Distributed cloud***, and ***network embedded cloud*** . A centralized cloud usually consists of one huge data center, while a distributed

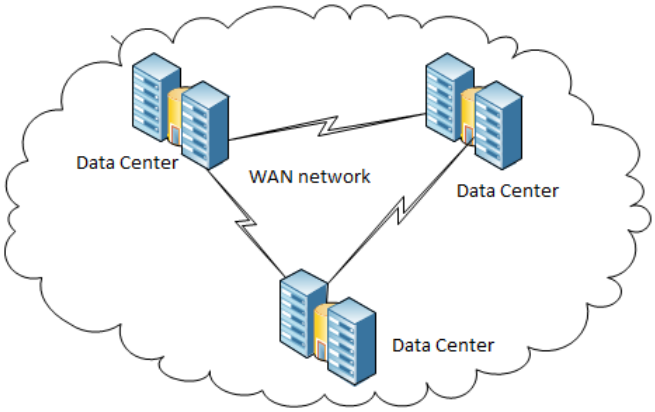

Figure 1.1: Basic Distributed cloud topology.

cloud usually consists of multiple geographically dispersed medium or small size data centers. These data centers are interconnected with customers (and each other) using wide area network (WAN) links to serve user requests over the public internet as shown in Figure 1.1.

The concept of a "*network embedded cloud*", also known as "***network enabled cloud***" or "***carrier cloud***", is not yet fully defined. This class of cloud takes cloud computing one step further than a "*distributed cloud*". In this approach network providers add compute resources to their existing network infrastructure in order to offer cloud services, while controlling these computing resources *and* controlling the network infrastructure. A network provider can offer cloud services based upon a large number of small size data centers, which are embedded in their carrier network, as shown in Figure 1.2. In this Figure "PN" stands for "process node" and depict the computational capability for network devices (i.e., small size data center beside router).

A cloud network typically has two parts: "*Compute Resources*", which provide an host for the requested resources* and a "*Cloud Management System*" (CMS). The CMS aims to provide secure, optimal, and scalable management of the cloud's resources. This gives the CMS broad responsibilities. "*Resource Allocation*" is one of the main responsibilities of the CMS. In order to achieve efficient resource allocation, the CMS needs to have sufficient information about

*The requested resources can be in the form of physical or virtual computing, storage, and networking resources.

the status of the cloud network (and its embedded computing resources if applicable). A "*Resource Monitoring*" system is needed in order to provide this information. The resource monitoring system provides information about all of the available resources in the cloud network, thus helping the resource allocation and resource management system to make the appropriate decisions.

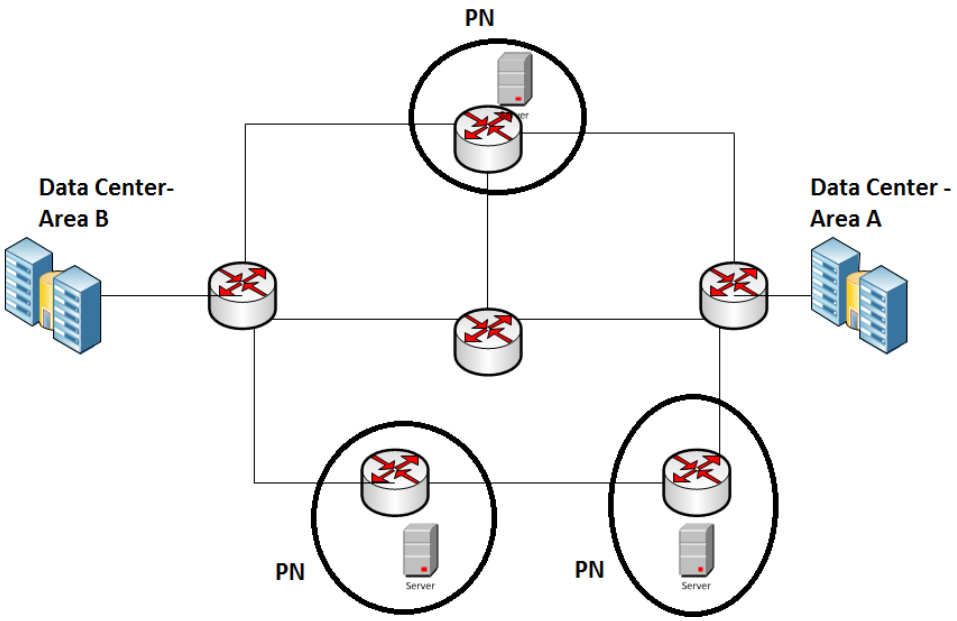

Figure 1.2: Network embedded cloud architecture. The PN refers to a process node and depicts a router with computational capability.

1.2 Problem description

As discussed above, resource allocation and resource monitoring play a vital role in a network embedded cloud. Resource allocation and resource monitoring in a centralized cloud architecture is a well understood problem with several commercial solutions, such as *Amazon web Services* [4], *Windows Azure* [5], and *Google App Engine* [6]. However, resource allocation and resource monitoring are more complex and challenging in a distributed cloud than in a centralized cloud; due to the larger number of both choices and constraints (e.g., geographic location preference, cost limits, etc.) that must be taken into consideration.

A network embedded cloud is an appealing concept for telecommunication operators because it can offer a potentially large business opportunity. Therefore, the major telecommunication operators and telecommunication vendors are competing for leadership in network embedded clouds [3]. A network embedded cloud provider needs advanced resource management, allocation, and monitoring in order to achieve high performance while providing high quality cloud services. They need to be able to provide these services efficiently in order to compete against their competitors.

It is important emphasizing that resource allocation and resource monitoring are much more complex and challenging for a network embedded cloud than for a distributed cloud, since the carrier's network infrastructure is also cloud resource. As a result, the resource monitoring system needs to provide information about the available resources at each of the data centers (e.g., CPU, RAM, storage, or different classes of virtual machines (VMs)) *and* information about the carrier networks (e.g., network topology and link attributes). As a result, in a *network embedded cloud* more constraints (e.g., network distance, latency bounds, and delay) need to be taken into account than in the case of a distributed cloud. In conclusion, in a network embedded cloud *network awareness* is essential in order to have efficient and optimal cloud management and resource allocation.

Due to the necessity of a resource monitoring system in a network embedded cloud and the lack of solutions for this problem in the literature, this thesis aims to provide a resource monitoring solution for a network embedded cloud.

1.3 Solution proposal

As discussed in section 1.2, the CMS requires information not only about reserveable capacity in each distributed data center, but also about network conditions including the topology and links characteristics. To provide these details two potential solutions are discussed briefly in this section.

The first alternative uses the traditional ways of monitoring the distributed data center resources [7], but creates an extension on it to be able to track link and network conditions. This solution is not feasible because the data centers are not directly connected to each other and we cannot monitor the links within the operator's network. As an example, the Transmission Control Protocol (TCP) connections from processes that are monitoring the data center's resources can be used by a CMS to collect the required information. In this case, we need to find a way to provide topology and links condition information, however this is

not easy. To collect this network related information, the information provided by a link state routing protocol can be utilized. Moreover, when the cloud provider needs a distributed management system this solution seems inefficient because each management system will need to collected the information. As a result, developing and implementing a reliable flooding mechanism is required or the management systems have to synchronize (i.e., exchange) their information. As link-state protocols already provide a solution for collecting and distributing topology and link state information, extending such a link-state protocol is another potential solution.

The second alternative uses a link-state routing protocol to monitor the network conditions and extend it in such a way that it also provides resource information of the distributed data centers. Link-state routing protocols can be classified in to two categories: *pull-based* and *push-based*.

In *pull-based* protocols, such as Simple Network Management Protocol (SNMP) [8], network information is retrieved on-demand. As a result, this approach is not scalable in a distributed environment, as CMS needs to explicitly ask all nodes to provide their updated information. Cloud and network resources are highly dynamic in nature, and the cloud provider receives many requests, then the cloud management system needs to frequently pull all the nodes in the network to have learn the current cloud network state. This situation can cause excessive network traffic and it also delays operations in the CMS, as the CMS must wait to get the information it has requested. It is worth mentioning that if the cloud providers reserve some amount of resources as surplus resources* then it is possible to greatly reduce the traffic - by taking an optimistic resource allocation approach.

When using a *push-based* link-state routing protocol, such as Open Shortest Path First (OSPF) [9], each node will send a link-state update to other nodes if required due to a state change. In this work, we consider such a push-base protocol in our solution. This thesis aims to provide a resource monitoring solution for a network embedded cloud by proposing an extension to the Open Shortest Path First-Traffic Engineering (OSPF-TE) protocols. It is worth highlighting that OSPF and OSPF-TE is already used widely by network operators [10, 11, 12], who are potential network embedded cloud providers, to manage their Multiprotocol Label Switching (MPLS) network. More information about push-base link-state protocols, particularly OSPF and OSPF-TE, are provided in Chapter 2.

*Note that if there are not surplus resources, then the system is likely to be unacceptable for the users at some points in time.

1.4 Goals

The goals of this master's thesis project are:

- Investigate the current state-of-the-art for cloud networking, with a focus on "*network embedded clouds*".
- Understand how OSPF-TE works and how it can be extended to support resource management in a network embedded cloud.
- Investigate how to design and implement extensions to OSPF-TE inside an OSPF router.
- Integrate the results into the Quagga router (As the initial target router is Quagga [13]).
- Evaluate the solution in terms of suitability, flexibility, performance, and robustness.

1.5 Methodology

In order to fulfill the objectives of this project, quantative and qualitative approaches were used. The research methodology was based on the agile project methodology, which meant that at each step, it was possible to develop and refine the scope of the project [14]. The problems in this thesis project were addressed in three phases as follows;

Literature study

In this phase, related work and required background material were studied. The resulting literature study described cloud computing in general, and especially the concept of a network embedded cloud. This step provided the essential background knowledge necessa-ry to obtain a state-of-the-art overview of the subject, and to understand the resource monitoring demands in a network embedded cloud. Different link state routing algorithms were considered with a focus on OSPF and OSPF-TE. The goals of this sub phase were: First, to understand OSPF and OSPF-TE behavior and LSA types and architecture of opaque

	LSA. Second, identify relevant processing metrics and virtualization information which OSPF-TE opaque LSAs need to convey. Third, investigate how OSP-TE can be extend- ed to convey cloud resource information.
Prototype implementation	Using the information from the previous phase, this phase aims to find and deploy a solution for resource monitoring in a network embedded cloud. The resource monitoring should provide information about link attributes and network topology, plus the relevant computing metrics of each embedded data center.
Evaluation	This phase examines the feasibility, suitability, flexibility, and performance of the proposed solutions(s). The focus of the evaluation is on protocol overhead. This phase starts with a theoretical analysis and calculating the expected protocol overhead. After that, some test scenarios were deployed in order to collected experimental data.

1.6 Limitations

The lack of background knowledge regarding a cloud network, the solution proposed in this thesis, and my limited programming skills were the first limitations. As the concept of a network embedded cloud is new and such clouds had not be deployed at the time that this thesis project started, there was little information available to the public in this area, and in particular there was no related work concerning resource monitoring of a network embedded cloud. Also, one and half months after this project started one of industrial supervisors for this project became sick and another industrial supervisor left the company.

Furthermore, the Quagga application, which was required to be part of solution, had some bugs. Those bugs are reported to the Quagga community and were fixed by the author of this thesis. Unfortunately, the Quagga website provided little information about the Quagga OSPF application programming interface (API). Additionally, no published documents were available for a Quagga developer. As a result, lots of time was spend to reviewing and studying the Quagga source code to gain knowledge about this API. Moreover, the source code of the Quagga router was not well commented which made this task harder.

After a while the author recognized that in order to implement the proposed solution it was necessary to choose and develop an update policy which was out of initial scope of this thesis. Subsequently, during testing we recognized that a data center model was required in order to be able to evaluate the performance of any solution. As a result, two unplanned tasks were add to the scope of this project.

As a result of the difficulties described above last step of the project, i.e., testing the performance of the extended OSPF-TE, was not completed. The author was unable to apply this solution in a real environment. As a result, testing was performed in a simulated environment with the goals of showing that the solution would function properly, and to generate some statistics for a subsequent performance evaluation. The results of this last step is biased due to our simple test bed topology, selection and implementation of a data center model, and update policy. Also, the test computer was unable to support more than six simulated Quagga routers and the data center model application. Therefore, the author was unable to create any complex scenarios.

Overcoming the limitations of this project leads to the future work proposed in section 7.2.

1.7 Thesis outline

This thesis is organized as follows:

- ***"Chapter 2"*** provides an overview of the subjects necessary to understand the rest of this thesis project.
- ***"Chapter 3"*** starts with a generalized solution to the problems presented in section 1.2. This is followed by details of the design of a proposed implementation. Also in this chapter, required considerations for the design are described.
- ***"Chapter 4"*** describes the implementation of the proposed solution and a data center model.
- ***"Chapter 5"*** describes different link-state update policies and contains experimental analysis on the behavior of those policies based on a proposed data center model in chapter 4.
- ***"Chapter 6"*** starts with a theoretic analysis of the protocol overhead of the extended OSPF-TE. This is followed by a description of three test

scenarios. The chapter concludes with a discussion of the test results that were obtained in a simulated test environment.

- ***Chapter 7*** summarizes the work and draws some conclusions. Additionally some suggestions for future work are made.

Chapter 2

Background study

This chapter provides background for the reader and introduces the concepts that will be utilized in the remainder of this thesis. The chapter begins by reviewing cloud computing theory and methodologies. Since we are proposing to use an extenstion to the open shortest path first (OSPF) protocol, we introduce the basic concepts of routing protocols, and then focus on OSPF and some of its extensions. Finally, some information about link-state update policy and available Open Source Routing (OSR) implementations, which are detailed in the following three chapters, are mentioned.

2.1 Cloud computing

The idea of cloud computing is not new. It has been around for years, in fact it was the driving force for the original ARPAnet and the subsequent "*Internet*" [15]. A typical cloud consists of one centralized or multiple geo-diverse data centers which are interconnected using WAN links. By utilizing this cloud, customers can ask for and receive services without needing to know exactly where these services are situated [15].

The goal of cloud computing is to shift the computing infrastructure (both hardware and software resources) into a logical cloud in order to reduce the cost of management and operation of these infrastructures. Cloud computing is an evolving paradigm, hence there are many definitions for it in the literature [16]. According to The United States National Institute for Standard Technology (NIST) [17], "*Cloud Computing*" is a model (**not** a technology), which consists of five characteristics, four deployment models, and three distinct service models as shown in Figure 2.1.

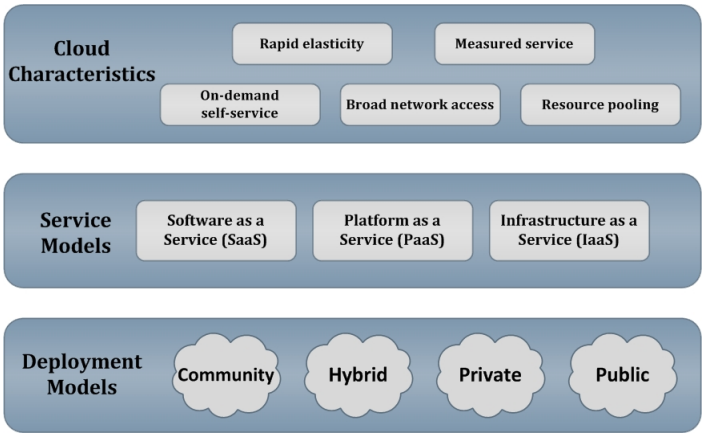

Figure 2.1: The NIST Cloud computing definitions.

2.1.1 Cloud characteristics

The five characteristics of cloud computing are:

On-demand self-service Based on this characteristic, the computing resources are provided automatically without requiring human interaction whenever a customer makes a request.

Broad network access Based on this characteristic, the computing resources can be accessed over the network with standard mechanisms. This feature provides platform-independent access for all types of clients (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling Based on this characteristic, the computing resources are shared. This means that multiple customers can use the same set of resources at the same time. The cloud provider's compute resources are pooled to serve multiple consumers using a multi-tenant model [18]. Physical and virtual resources are dynamically allocated or reallocated according to customer demand. Location independency is an inherent part of pooling. In fact, the location of resources (e.g., virtual machine, processing, storage, memory, and network bandwidth) is generally hidden from the cloud's users. It is worth mentioning that although the customers do not have control or knowledge of the exact location of a resource, they may

be able to identify the location at a higher level (e.g., country, state, or data center). This location may be part of their requirements, for example in order to meet legal requirements regarding data protection.

Rapid elasticity Based on this characteristic, the computing resources should appear to the customer to be infinite and available in any quantity at any time. To achieve this goal, the cloud provider has to be able to scale computing resources both up and down as needed. This could happen automatically (e.g., by turning on/off some racks in a data center) or manually (e.g., adding some additional infrastructure such as servers and racks into the data centers).

Measured service Based on this characteristic, cloud computing resource usage can be controlled, monitored, and reported based on service type (e.g., storage, processing, bandwidth, and active user accounts). Typically, this is done according to a pay-per-use or charge-per-use policy.

2.1.2 Cloud service models

As discussed above, cloud providers strive to provide different services over the cloud to their customers, however with respect to the Everything as a Service (*XaaS*) [1] model, these services are **not** limited by NIST's definition [19]. As defined by NIST, the basic service models are:

Software as a Service (SaaS) In the SaaS model, applications are hosted by cloud providers. A customer can take advantage of an application through a user interfaces (e.g., web browsers or program interface) "*anytime, anywhere*". In this service model, the cloud user is responsible for managing and entering their own data, while everything below the application is the cloud provider's responsibility.

Platform as a Service (PaaS) In the PaaS model, the platforms* and compute

*Virtual machines, operating systems, and development framework can be considered as a platform.

capability are hosted by cloud providers. As a result, the customer can deploy and utilize their own software* on the cloud's platforms. In this service model, the client is responsible for installing and managing their own application, while the cloud provider is responsible for everything from the platform down, such as managing the cloud infrastructure and the operating system(s).

Infrastructure as a Service (IaaS) In the IaaS model, the computing resources (e.g., servers, storage, and network) are hosted by cloud providers. The cloud provider is only responsible for managing the infrastructure, while everything above this infrastructure is the customer's responsibility (e.g., the operating system (OS), application, and user interaction).

2.1.3 Cloud Deployment Models

As defined by NIST, there are four main cloud deployment models:

Private Cloud In this model, the cloud is operated exclusively for a single organization. This organization may own and manage the cloud by itself, the cloud can be owned by a third party, or any combination of these. A private cloud may be on-promises or not.

Community Cloud In this model, the cloud is operated for the exclusive use a community of customers. These customers can be from the same organization or independent organizations, but with similar concerns with regard to policies, security, and mission. A community cloud may be owned, managed, and operated by one or more of the organizations, a third party, or any combination of these. A community cloud may be on-promises or not.

*This software can be developed by the customer themselves or be acquired from a third party.

Public Cloud In this model, the cloud is operated for public use. The cloud may be owned, managed, and operated by an organization such as a business, an academic organization, or a government.

Hybrid Cloud In this model, the cloud is composed of two or more different cloud models (i.e., private, public, or community cloud). These clouds preserve their characteristics, but the clouds are bounded together as one unit.

2.1.4 Varieties of Cloud

The idea behind cloud computing is to centralized as much functionality as possible. As a result, current cloud providers typically build extremely large data centers in order to offer low cost services to their customers. This class of cloud architecture is known as a "***centralized cloud***". In this case, the cloud's services can be accessed via the internet and WAN links. Building such data centers is extremely expensive with respect to the cost of servers, power, cooling, networking, physical plant, etc. [20]. In addition, a centralized cloud inherently increases the average distance to the end users, which many consider to be a negative attribute of this class of cloud.

2.1.4.1 Distributed Cloud

Unlike the centralized cloud, a ***"Distributed Cloud***" is based upon a number of large data centers located at a number of different locations [21]. These distributed data centers are connected both to each other *and* to the network at their location. Distributed clouds have similar characteristics to centralized clouds, but offer some additional benefits. The geo-diversity of smaller data centers can be more attractive since they are less expensive to build and manage [20]. In addition, a distributed cloud architecture moves services closer to end users, hence providing low latency, while reducing the network capacity needed by each data center.

Note that there those such as Bill St. Arnaud, former Chief Research Officer for Canada's Advanced Internet Development Organization (CANARIE), who believe that these computing centers should be located where power and cooling are inexpensive and that the bits can efficiently be moved via fiber networks to where the users are, thus achieving a more cost effective and more "green" information technology infrastructure.

2.1.4.2 Carrier network and cloud

Although a distributed cloud reduces latency for end users and reduces the required bandwidth capacity for data centers, it still has considerable shortcomings [22]. Fundamentally the use of a cloud assumes that the network, which connects the cloud providers and clients, meets all the requirements of the client-cloud interactions. However, cloud providers usually do not own the network(s) that their cloud services travel over, as this is typically some carrier's network (or multiple carriers' networks). As a result, the cloud provider does not have control over the quality of experience (QoE) of the end user beyond the network within their data centers. Therefore, the cloud provider cannot guarantee end-to-end performance, if the network is a bottleneck. The solution to this problem is generally based upon one or more Service Level Agreements (SLAs) between the carrier and the cloud provider and between each of the end users and their local internet service providers (ISPs). The need for SLAs and questions about security are key reasons cited by many businesses and organizations that utilize latency-sensitive applications (such as telecommunication application, video conferencing, online gaming, or businesses that use Java script and XML) for why they hesitate to migrate into clouds.

Today cloud providers try to make agreements with their network providers to overcome the shortcomings discussed above due to the network in order to gain advantage over their competitors. This solution may not always be feasible, especially for mobile networks. Unfortunately networks, especially mobile networks, have capacity limitations and upgrading the network is not always feasible due to the required capital cost. These limitations (of the network) and questions about security provide an opportunity for a new cloud paradigm referred to as a ***"network embedded cloud"***.

2.1.4.3 Network embedded cloud

A ***"network embedded cloud"***, also known as a ***"carrier cloud"*** or ***"network enabled cloud"***, has a distributed cloud architecture where the computing resources are embedded in, and distributed across the carrier's network. This idea is based upon on a transformation of carrier-grade networks, which belong to cloud providers or telecommunication operators, to cloud computing. In this model the network resources are simply another set of cloud resources. As a result all of the computing and network resources are viewed as being elastic and allocated based upon demand. The concept of a network embedded cloud is appealing for network providers, since they can enter the cloud market and increase their

revenue by adding cloud functionality to their existing access networks [23].

The concept of a network embedded cloud is an emerging and evolving concept in cloud networking, and it is not yet fully defined. However, many large vendors (including Alcatel-Lucent, Cisco Systems Inc, Ericsson AB, Huawei Technologies Co. Ltd., and Nokia Siemens Networks) are competing for network embedded cloud leadership [3]. As an example, Ericsson AB announced their network embedded cloud concept at the Mobile World Congress 2013 [24] and IBM announced its cloud strategies for virtualization of a telecommunication infrastructure [25].

A network embedded cloud solves some of the shortcomings of cloud networking, while retaining all of fundamental benefits of a cloud [26, 27]. network embedded cloud providers can offer end-to-end cloud services and can also guarantee QoE by integrating their network into their cloud. In this class of cloud architecture, both computing resources and network infrastructures are operated by the cloud providers (i.e., a carrier), hence they can reserve network resources and allocate compute resources on demand. In a network embedded cloud, latency-sensitive and delay-sensitive services can be provided very close to the users, hence these services can meet low minimum delay requirements. In addition, the aggregate bandwidth requirements for these services will be reduced as the traffic will flow along a shorter path.

However, there are questions of just how closely these resources need to be provisioned and whether existing networks actually have bottlenecks that prevent the other cloud models from providing adequate service and acceptable QoE. Moreover, there are a variety of solutions to the security problems when utilizing remote cloud services, hence it is not clear that a network embedded cloud has any advantage with respect to security.

2.1.4.4 Cloud Management System

In order to achieve optimal and high performance cloud services, a cloud network must be managed properly. A "***Cloud Management System (CMS)***" is responsible for management of the cloud's network. Cloud management includes several tasks, such as operating and administrating the cloud, capacity planning, performance monitoring, security management, resource management, and virtualization. [28, chapter 11].

One of the key functions performed by the CMS is ***"Resource Allocation"***. Resource allocation mainly focuses on allocating the resources required to satisfy

user demands. In a centralized cloud architecture, resource allocation is limited to finding the best location within the data center to instantiate a VM(s) to fulfill the user's requirements. This selection can be done based on various policies, such as best effort, energy-awareness, cost, performance, or any combination of these policies. Resource allocation has been the subject of several many papers in recent years [29] and has been a classic issue in operating systems.

In the case of a distributed cloud architecture, due to its geo-diversity, resource allocation faces similar, but more advanced challenges and requirements as compared to a traditional centralized cloud architecture. The resource allocation challenges in a distributed cloud can be addressed by resource modeling, resource offering and treatment, resource discovery and monitoring, and resource selection [21]. In the other words, distributed cloud providers need to do a number of tasks in order to serve requests. First, they need to model their resources based on the services that they are going to offer. Second, when a service request arrives, the cloud providers need to be aware of the current status of their cloud's network in order to determine whether there are suitable available resources to satisfy the request or not. Finally, if there are suitable available resources, then the cloud provider needs to allocate these resources in the best manner to serve the service request, while continuing to support existing allocations and preparing to handle future requests.

A ***"Resource Monitoring"*** system is a key element in a network embedded cloud. As mentioned earlier in a network embedded cloud the communications network assets are parts of the cloud architecture, so efficient resource allocation and resource placement requires *network awareness*. In fact, because the network embedded cloud provider has information about both network *and* data centers it can allocate services based on more constraints, including operational cost, location, SLAs, and data center availability.

Currently there is very little literature available regarding resource monitoring techniques for a network embedded cloud. As discussed in section 1.3, there are some proposals for resource monitoring in a network embedded cloud. In this thesis project we propose an extension to OSPF-TE for resource monitoring for use with this class of cloud network. The information that will be collected will include network topology and the link characteristics (e.g., delay, bandwidth, and jitter), location of compute nodes, available compute resources (e.g., CPU, RAM, and storage) in each of the data centers. This information can be used by a network embedded cloud's CMS to allocate resources in order to satisfy user demands.

2.2 Routing protocols

The purpose of a communication network in the simplest case is to connect two end points. These two end points could be far from each other. As a result, data flowing between the source and destination could pass through multiple intermediate devices, i.e., routers. There are two ways of routing: static routing and dynamic routing. In static routing, the network administrator is responsible for manually specifying routing paths. In contrast, in dynamic routing the routers exchange information in order to dynamically set up routing paths. The routers exchange routing information and apply a routing algorithm. Distance-Vector (DV) routing algorithms and Link-State (LS) routing algorithms are two common classes of algorithms used by dynamic routing protocols [30].

2.2.1 Distance-Vector routing algorithms

A device using a DV routing algorithm informs its directly attached neighbors periodically of its entire routing table. The recipient routers can use any DV algorithm* to update their routing table, and then they distribute these routes to their neighbors. In this class of algorithms, devices describe the network in terms of adjacent neighbors and each router does **not** have knowledge of the network's complete topology. The Routing Information Protocol (RIP) [31] and Interior Gateway Routing Protocol (IGRP) [32] are examples of DV routing protocols.

2.2.2 Link-State routing algorithms

A device using a LS routing algorithm, informs all other devices in the network of its connectivity by broadcasting its LS information. As a result of this broadcast, each router will have an identical view of the complete topology of the network, hence it can construct its own connectivity map. Finally, each device can use any LS algorithm† to compute the best logical path from itself as a source to every other node in the network as a destination. Open Shortest Path First (OSPF) [33] and Intermediate System to Intermediate System (IS-IS) [34] are examples of the Link-State routing protocols.

*Bellman-Ford algorithm, Ford-Fulkerson algorithm, and DUAL FSM are examples of DV algorithms.

†Dijsktra and Prim's algorithm are examples of LS algorithms.

2.3 Open Shortest Path First

The Open Shortest Path First (OSPF) [33] protocol is a link-state routing protocol. It is classified as an Interior Gateway Protocol (IGP). In an OSPF network, routing information (e.g., the set of usable interfaces and neighbors) will be encapsulated in a unit of data referred to as a Link-State Advertisement (LSA). The router floods its LSAs throughout the Autonomous System (AS)*. The participating OSPF routers in an AS will each keep the received LSA data in a database, which is referring to as a Link-State Database (LSDB). As a result of the flooding of LSAs, all the OSPF routers within the AS will have the same contents in their LSDBs. This information describes the AS's topology. Each OSPF router computes a Shortest Path First (SPF) tree, using Dijkstra's Algorithm [35] using the information in its LSDB. The SPF tree is used to populate the router's routing table.

OSPF is a dynamic routing protocol and it can quickly detect a topology change. In the event of a topological change, the OSPF router notifies other routers by flooding new LSAs (Also known as Link-State Update (LSU)). The other OSPF routers in the AS will update their LSDBs and recalculate their routing table when they receive topology update notifications.

The OSPF protocol allows a set of networks and hosts to be grouped together. Such a group is called an *area*. The routers within an area are called intra-area routers. The intra-area routers have identical topological information, and this topology is hidden from the other ASs' areas. The OSPF areas are interconnected via a backbone area which is referred to as area zero. A router located on the border of an OSPF area, interconnects the area(s) to the backbone area. This router is called a Area Border Router (ABR). The ABR keeps separate topological information for each area. The OSPF protocol runs directly over the Internet Protocol (IP) and has five different message types to accomplish its operation. These message types are shown in Table 2.1. Several of the OSPF message types LSAs are responsible for passing topological information.

*A group of routers with a common administration who exchanging routing information with the same routing protocol.

Table 2.1: OSPF message types.

Type	Message Name	Protocol Function	Description
1	Hello	Discover/maintain neighbors	Router uses this message to discover adjacent routers and establish relationships between neighboring devices.
2	Database Description	Summarize database contents	These packets are exchanged when an adjacency is being initialized. These messages describe the contents of the topological database.
3	LS Request	Database download	Link-state request packets are used to request a portion of the LSDB from another router. This will send when the router notify parts of its LSDB are out of date.
4	LS Update (LSU)	Database update	These messages are sent in response to a link-state request. They carry a collection of link-state advertisements routers uses these information to update the information in the LSDBs of routers that receive them.
5	LS ACK	Flooding ACKs	These messages aims to provide reliable link-state exchange process, by acknowledging (ACK)receipt LSU.

2.3.1 OSPF operations

When an OSPF router starts up, it first checks its directly connected links and networks and then detects which of these participate in the OSPF routing process. In this step an OSPF router tries to create a LSA based upon the information about its directly connected links, such as the interface's IP address, link cost(s), and network type. Once the OSPF router has determined which interfaces belong to the OSPF routing process, the OSPF router attempts to discover its neighbors using a OSPF "*Hello*" message. If an OSPF router receives a Hello message in response, then it forms an *adjacency*. An adjacency is a relationship with a neighboring router. After a adjacency relation established, both routers will periodically send Hello messages to ensure that the neighboring router is alive. As a third step, the OSPF router builds its LSA containing the link-state information (such as neighbor routers and links). After building the LSA, the OSPF router floods its LSA into the OSPF domain. As a result, all other OSPF routers in a OSPF domain will receive this LSA. Finally, as a fifth step, the OSPF routers construct their LSDB using the information from the LSAs. This LSDB is subsequently used for calculating a SPF tree and routing table.

It is worth highlighting that, a OSPF router floods LSAs in two cases. Case one, when the OSPF process starts; and case two, when the router state changes (e.g., link failure, link recovery, and neighbor going down). When the OSPF router detects a link-state change, it will create a new LSA containing information about only the particular network or link that has changed, and then it floods this new LSA to inform others routers about that change, see section 2.4. Other OSPF routers in the OSPF domain will update their LSDB and recalculate their routing table when they receive topology change notifications.

2.3.2 Link-State Advertisements

The LSAs are the basic unit of data which are used to convey information about the local state of a router. Each LSA begins with a standard 20-byte header as shown in Figure 2.2.

The LSA header contains information to identify the link as the combination of LSA type, link-state ID, and Advertising Router fields. These fields in the LSA header represent a unique identification for each LSA. The OSPF router can originate one or more types of LSAs. Eleven distinct types of LSA are registered for the OSPF protocol [36]. These are listed in Table 2.2.

Table 2.2: OSPF LSA types.

Value	LSA type	Description	Reference
1	Router-LSA	Describe how an area's routers and networks are interconnected	RFC 2328 [33]
2	Network-LSA	Describe how an area's routers and networks are interconnected	RFC 2328 [33]
3	Summary-LSA (IP network)	When area are used it provide information about network	RFC 2328 [33]
4	Summary-LSA (ASBR)	When area are used it provide information about links and AS boundary router	RFC 2328 [33]
5	AS-external-LSA	It provide information about external links outside the AS	RFC 2328 [33]
6	Group-membership-LSA	This type is defined for Multicast Open Shortest Path First (MOSPF) which is an extention to OSPF	RFC 1584 [37]
7	NSSA AS external LSA	Not-so-stubby-area (NSSA) routers use this type to provide their external routes information for ABRs	RFC 3101 [38]
8	Reserved	-	-
9	Link-local Opaque LSA	This LSA conveys Opaque data. The Link-local Opaque LSA is limited to a single link and it never forwarded to other links by Recipient router.	RFC 2370 [39]
10	Area-local Opaque LSA	This LSA Conveys Opaque data. The Area-local Opaque LSA is limited to a single area which means it flooded by all the router with in an area but it is never forwarded by an ABR to other areas.	RFC 2370 [39]
11	AS Opaque LSA	This LSA Conveys Opaque data. The Type 11 Opaque LSA is flooded throughout the OSPF domain and all routers in all Areas will receive this LSA.	RFC 2370 [39]

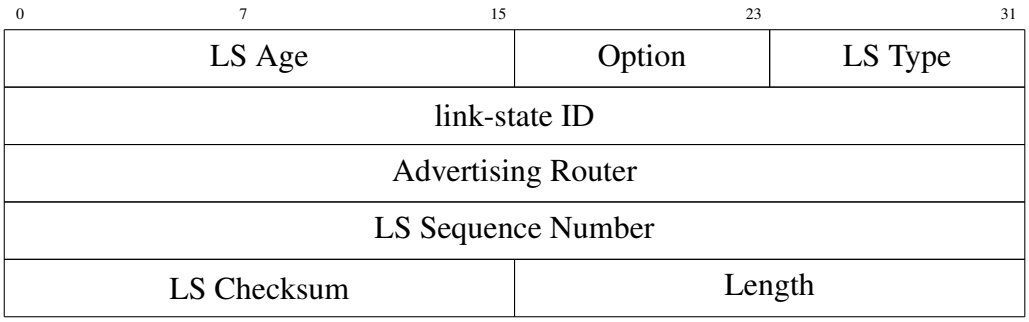

Figure 2.2: Common LSA Header Format.

The LSA type 1, LSA type 2, LSA type 3, LSA type 4, and LSA type 5 are an essential part of OSPF version 2 protocols and are necessary for the basic functionality of the OSPF protocol. The remaining types were added later to the OSPF standard in order to add new capabilities to the OSPF protocol.

The "***Router LSA***" (Type 1) is perhaps the most influential LSA type in the OSPF protocol. This LSA type describes router interfaces in such a way that other routers in the same area can use it to build a network topology and SPF tree. This information travels through the area without any modification and all the routers have the same copy of this information.

The LSA header is followed by a LSA body. The information in the LSA body varies based on LSA type. The body of a "*Router LSA*" contains information about the state and cost of a router's links and details about the router. In fact, the "*Router LSA*" provides most of the required topological information. The "*Network LSA*" body includes a sub-net mask and information about all routers on the network. The "*Summary LSAs*" (LSA type 3 and 4) includes metrics and summarized addresses. Finally, the AS-External LSA body has some fields to allow the exterior router to be reachable. For a more detailed description of OSPF message formats see Appendix A of RFC 2328 [33]. Also, a brief description of LSA types is given in Table 2.2. The opaque LSAs are explored in the next subsection.

2.3.3 Opaque Link-State Advertisement

Opaque Link-State Advertisements (Opaque LSAs) [40] are types 9, 10, and 11 LSAs. These Opaque LSAs provide an effective way to spread arbitrary information using the OSPF protocols into OSPF areas. In fact, an Opaque LSA can piggyback **any** information and flood this information within an OSPF

network. The information contained in Opaque LSAs may be used by OSPF protocols or any other application in the OSPF domain that can utilized OSPF as a transport protocol. The three types of Opaque LSA differ only in their flooding scope as shown in last three rows of Table 2.2.

An Opaque LSA has a standard LSA header, just as another OSPF LSAs, but with minor differences in the link-state ID structure. The link-state ID syntax for an Opaque LSA consists of two parts: "*Opaque type*" field (the first 8 bits) and a "*Opaque ID*" (the remaining 24 bits). The packet format of an Opaque LSA is shown in Figure 2.3.

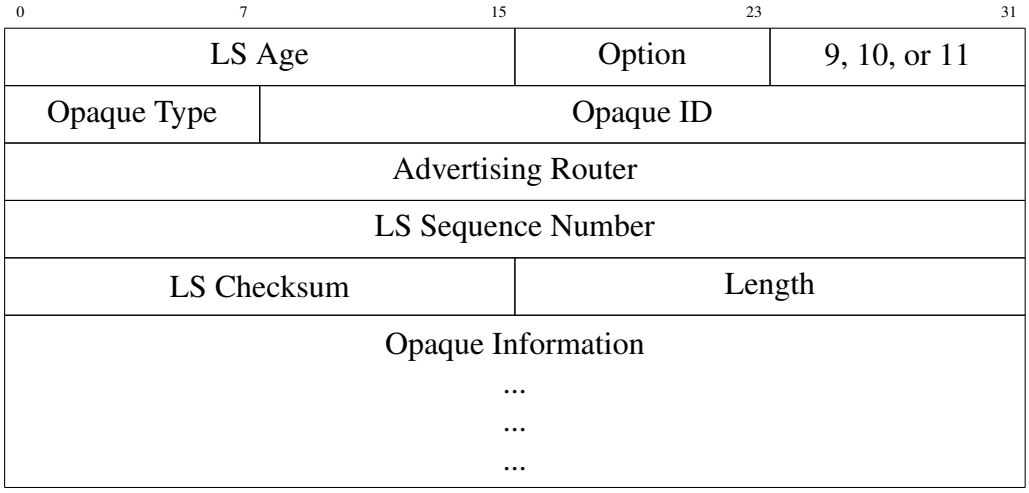

Figure 2.3: Opaque LSA Header Format.

The Opaque Type field identifies the application of the Opaque LSA. As Table 2.3 shows, six type values have been allocated by the Internet Assigned Numbers Authority (IANA) through the OSPF working group [41]. In addition, it is important to highlight that the combination of the Opaque ID and Opaque Type form a unique identifier for this specific type of LSA.

Table 2.3: Different "*Opaque Type*" values in Opaque header.

Value	Opaque Type	Description	References
1	Traffic Engineering LSA	Used for MPLS-TE.	RFC 3630 [42]
2	Sycamore-Optical Topology Descriptions	Used to communicate details of optical devices.	John Moy [41]
3	Grace LSA	Used for OSPF hitless restart.	RFC 3623 [43]
4	Router-Information LSA	Used for advertising optional capabilities.	RFC 4970 [44]
5	L1VPN LSA	Provide OSPF-based L1VPN auto-discovery.	RFC 5252 [45]
6	Inter-AS-TE-v2 LSA	For the advertisement of OSPFv2 inter-AS TE links.	RFC 5392 [46]
7-127	Unassigned	Can be allocated by the IANA through the OSPF working group for future Opaque LSA types.	
128-255	Reserved	Used for private and experimental use.	RFC 5250 [40]

2.3.4 OSPF Traffic Engineering

The Traffic Engineering (TE) Extensions to OSPF, also known as OSPF-TE [42], is an extension to the OSPF protocol to add traffic engineering capabilities to it. OSPF-TE uses Opaque LSA type 10, which has an area flooding scope, to carry TE information.

The Opaque LSA type 10 is known as the "*Traffic Engineering LSA*" (TE-LSA). This LSA performs the same function as Router LSAs. In another words, the TE-LSA can identify the originating router, the router's neighbors, and TE parameters. This LSA can be used to build an *extended* LSDB, known as a "***Traffic Engineering Database***" (TED). The TED has some additional link attributes (e.g., bandwidth* and administrative constraints) compared to the LSDB.

It is worth highlighting that non-TE capable routers in an OSPF network can flood TE-LSAs, just as any other Opaque LSAs. Hence, an OSPF network can have both non-TE and TE capable routers, while still providing traffic engineering functionality. The TED is used for monitoring the extended link attributes, local constraint base source routing, and for general traffic engineering purposes. "***Constraint Base Routing***" computes a path from a source node to a destination node that satisfies a set of constraints [47, 48].

The Traffic Engineering LSA starts with common LSA header with LSA Type 10 and Opaque Type 1. The payload portion of the TE-LSA carries information in the format of one or more nested triplets: Type/Length/Value (TLV), as shown in Figure 2.4.

*These attributes can include maximum bandwidth, maximum reserveable bandwidth, unreserved bandwidth, or any combination of these attributes.

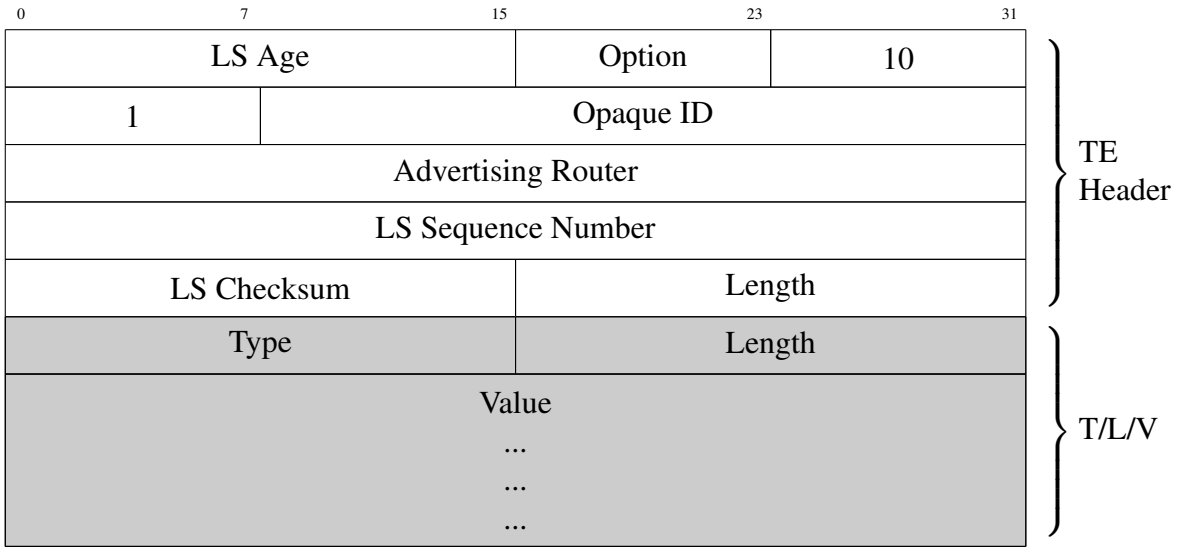

Figure 2.4: TE-LSA header format and TLV.

The "*Type*" field of TLV shows the type of the TLV and the "*Length*" field shows the length of the "*Value*" in octets. Table 2.4 shows the IANA registered top level TLVs for OSPF-TE [41].

Table 2.4: IANA registered top level TLV Types for TE LSA.

Value	**Top Level Types**	**Reference**
0	Reserved	RFC 3630 [42]
1	Router Address	RFC 3630 [42]
2	Link	RFC 3630 [42]
3	Router IPv6 Address	RFC 5329 [49]
4	Link Local	RFC 4203 [50]
5	Node Attribute	RFC 5786 [51]
6-32767	Unassigned	
32768-32777	Experimental use	RFC 3630 [42]
32778-65535	Reserved	RFC 3630 [42]

The "*Router Address*" TLV (Type = 1) specifies a stable IP address of the advertising router. It is worth mentioning that some TLVs in a TE-LSA are constructed of a series of sub-TLVs. IANA has assigned 25 sub-TLVs for the

Link TLV (Type value = 2) and two sub-TLVs for the Node Attribute TLV (Type value = 5). Table 2.5 shows the two IANA assigned sub TLVs for "*Node Attribute*" TLV [41].

Table 2.5: Types for sub-TLVs of a TE Node Attribute TLV (Value 5) as assigned by IANA

Value	**Sub-TLV**	**Reference**
0	Reserved	RFC 5786 [51]
1	Node IPv4 Local Address	RFC 5786 [51]
2	Node IPv6 Local Address	RFC 5786 [51]
3-32767	Unassigned	
32768-32777	Experimental use	RFC 5786 [51]
32778-65535	Reserved	RFC 5786 [51]

2.4 OSPF reliable flooding and flooding control

The OSPF protocol uses a reliable flooding mechanism to guarantee synchronization between routers' LSDBs. To achieve this goal, when a router starts, it needs to create and flood LSAs into the OSPF network. Also, the router has to regenerate its self-originated LSAs and send them to neighbors if the router's local states have changed.

When one of the router's neighbors receives a LSA, it should send an acknowledgment and examine the LSA's content. If the LSA contains more recent information that than in the router's LSDB, then the router updates its LSDB. Next, the recipient router forwards this LSA to all interfaces except the one that received that LSA. In this way, the OSPF protocol can ensure that all generated LSAs will be delivered to all the nodes within the AS.

OSPF's reliable flooding is robust in the face of errors which means the network operates correctly even if an error happens. To achieve this robustness LSAs are aged and include checksums. OSPF routers can examine the LSA Age and Checksum fields to detect corrupted or out of dated LSAs. Also, OSPF employs timers in order to control LSA flooding, as shown in Table 2.6.

Table 2.6: OSPF Timers

Timer	Default value	Definition
MinLSArrival	1 second	The minimum interval that a router can accept LSA.
MinLSInterval	5 second	The minimum interval that a router can generate LSA.
CheckAge	5 minutes	The interval that a router checks the checksum value.
LSRefreshTime	30 minutes	The maximum interval for re-originating any particular LSA.
MaxAge	1 hour	The maximum age that an LSA can accept in a routing table calculation.

The **MinLSInterval** and **MinLSArrival** timers limit the frequency of originating or accepting LSAs, respectively. The MinLSInterval timer specifies the minimum time interval during which an OSPF router can originate at most one LSA. The default value of MinLSInterval is set to 5 seconds. The MinLSArrival timer specifies the minimum time interval during which an OSPF router can accept a new LSA if a previous copy of this LSA already exists. The default value of MinLSArrival is set to 1 second, and LSA instances received at higher rates will discard. These two timers act as hold-down timers for the OSPF protocol.

Although MinLSInterval and MinLSArrival timers control the LSA flooding, a more advanced control mechanism is required to reduce unnecessary LSA updates when resources rapidly fluctuate (e.g. TE metrics or cloud resources change). The next subsection describe some proposed techniques to provide more advanced control of flooding.

Suppose an OSPF network with a full mesh topology which contains "N" routers. If one router in a network needs to send a LSA, it sends $(N-1)$* LSAs to all the router in the area. In response, each router other than the LSA originator router sends $(N-2)$ LSAs to all the routers except for itself and the LSA originator router. As a result, the total number of the LSAs which loads in an OSPF network by injection of one LSA can be calculated as shown in equation 2.1.

* All routers except itself.

$$Total\ number\ of\ redundant\ LSAs = (N-1) + (N-1)(N-2) = (N-1)^2. \quad (2.1)$$

As a conclusion, in a fully-connected mesh OSPF network with "N" routers each LSA requires an order of N^2 redundant messages to be flooded. This issue known as the *"LSA N-squared problem"* which can lead to scalability problems since the value of N could be very large [52]. To reduce the number of LSAs several techniques have been proposed including an *area* approach, a spanning-tree network architecture, and a configuration-information approach.

2.5 Link-state update policies

Link-state update policy concerns the methods of link-state update distribution within the network. As the routers or third party application use link-state information for making decisions, having up-to-date information is necessary to make an appropriate decision. As flooding link-state information can result in additional traffic, a good trade-off has to be found between the frequency of updates and the extra load on the network. Several link state update policies including Periodic, Immediate, Threshold-based, and Class-based update policy have been proposed in literature [53, 54, 55]. Some of these update policies are discussed below.

- **Periodic update policy:** In this technique, the router generates new LSAs based upon a predetermined period "T", thus after a period "T" the router floods LSAs into the network regardless of whether or not it has new information. In a periodic update, the fixed period "T" is the key factor sending updates and this period is independent of link-states. This policy has a high blocking probability and generates a lot of update overhead in realistic networks [53].

- **Immediate update policy:** When using this policy, the router floods LSA updates as soon as new information is available. This immediate update policy can be used for the resources which rarely change, but this policy has a bad effect (in terms of generating a lot of traffic) when the frequency of resources changes are high.

- **Threshold-based policy:** In the threshold-based policies, a new link-state update triggers when the available resources changes by a predefined threshold value. There are two categories for threshold-based policies: "*absolute threshold-based*" and "*relative threshold-based*" update policy.

In the **absolute threshold-based policy**, updates are disseminated when the absolute change between the current and the previously advertised value of a resource exceeds a certain threshold value (as Shown in equation 2.2) [56].

$$\frac{\left|R_n - R_p\right|}{C} \geq thr \tag{2.2}$$

Where R_n represents the current value of a resource, R_p denotes the previously flooded value, and C denotes the total capacity. With respect to this policy, the threshold value and the capacity are the key factors determining when updates are send. Therefore, a small threshold value and/or small amount capacity results in an increase in the number of updates when the resource changes a lot [57].

In the **relative threshold-based policy**, updates are disseminated when the relative change between the current and the previously advertised value of a resource exceeds a certain threshold value (as shown in equation 2.3) [56].

$$\frac{\left|R_n - R_p\right|}{R_p} \geq thr \tag{2.3}$$

Where R_n represents the current value of a resource and R_p denotes the previously flooded value. With respect to this policy, the threshold value is the key factor determining when updates are send. Therefore, a small threshold value results in an increase in the number of updates when the resource changes a lot [57]. In relative threshold-based policies, updates are sent more frequently when resources become low. This policy generates fewer updates than a periodic update policy.

- **Class-based policy:** In this technique, the resource is divided into classes and updates are flooded whenever the resource value move to a different class from the current class. When the maximum capacity of the resource is C, classes can described as follow:

$$\Big[0, \beta C\Big), \Big[\beta C, (f+1)\beta C\Big), \Big[(f+1)\beta C, (f^2+f+1)\beta C\Big), \ldots \tag{2.4}$$

Where "β" denotes a *base class factor* (β < 1), and "*f*" denotes a *class bindery factor*. The class's size could be fixed ($f = 1$) or exponential ($f > 1$). A class-based policy performs approximately the same as the relative change policy [58].

2.6 Open source routing suite

XORP [59], BIRD [60], and Quagga [13] are examples of software suites which provide open source routing (OSR). These OSR suites support many different routing protocols, including OSPF, RIP, and Border Gateway Protocol (BGP). XORP and Quagga provide an API to access to their OSPF demon. Developers can use this API to communicate and add additional functionality to routing suites without making any changes in the router's core source code.

2.6.1 Quagga router

Quagga is an OSR software suite developed for UNIX platforms. The Quagga router consists of Zebra, OSPF, RIP, and BGP demons (see Figure 2.5). The core demon Zebra is an abstraction layer of the underlying OS, while the OSPF, RIP, and BGP demons are responsible for providing routing functionality for these three different routing protocols.

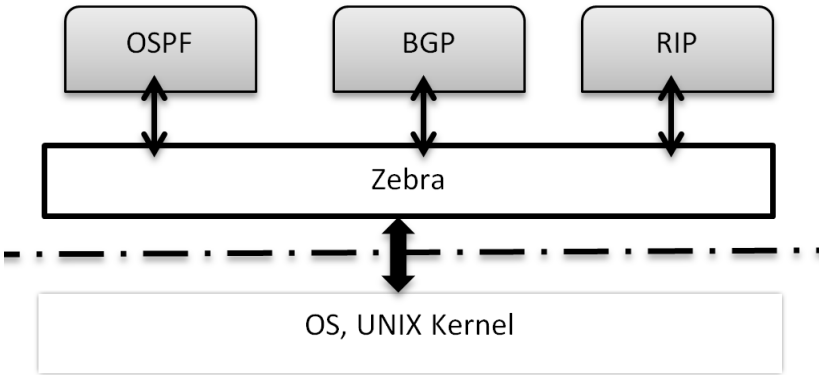

Figure 2.5: Quagga router system architecture.

The ***"OSPF daemon"*** consists of an OSPF core, opaque LSA, MPLS-TE, and OSPF API modules as shown in Figure 2.6. The *"OSPF core"* module is responsible for implementing the main tasks of the OSPF protocol (such as

neighbor discovery and exchanging neighbor state). The *"opaque LSA"* module enables the OSPF demon to exchange opaque LSAs with other OSPF routers. The *"MPLS-TE"* module and the *"OSPF API"* module can generate opaque LSAs and then invoke the *"opaque LSA"* module to flood the LSA into the OSPF domain.

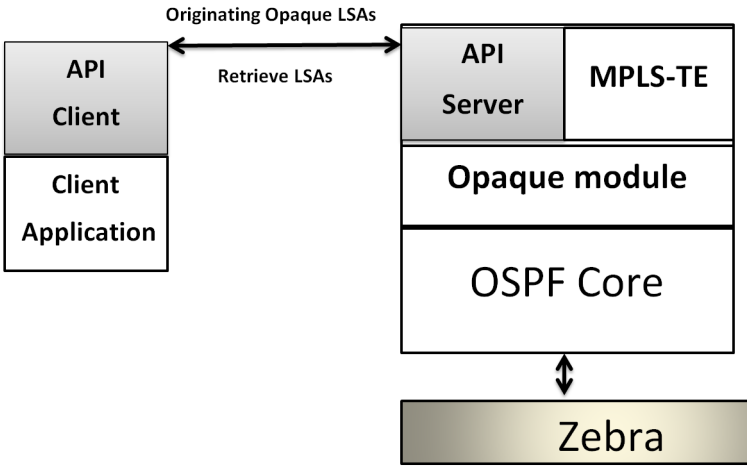

Figure 2.6: Quagga OSPF demon architecture.

2.6.2 Quagga OSPF API

The ***"OSPF API"*** consists of two parts: *OSPF API-Server* and *OSPF API-Client*. The *"OSPF API-Client"* can establish a TCP connection with the *"OSPF API-Server"* using ***"OSPF API Protocol"*** [61]. This connection can be utilized to retrieve the whole or part of Quagga router's LSDB from the Quagga router's OSPF daemon. Also, the OSPF API provides synchronization, which means that whenever a new LSA arrives at the OSPF daemon the OSPF API-Server immediately forwards this messages to the OSPF client application. In addition, the OSPF API-Client can generates an opaque LSA and ask the server side to flood it. The OSPF API protocol states are represented in follows and illustrated in Figure 2.7.

- **Initialize connection:** There are two connections between the OSPF API-Client and the OSPF API-Server. One connection is responsible for handling the *synchronous* messages, and another is responsible for handling the *asynchronous* messages. To establish these connection, first the OSPF API-Client uses the target router IP address with any port number to

begin the connection for synchronous messages. The OSPF API-Client application request/reply operates synchronously, which means each OSPF API-Client request message answered with a response message from a OSPF API-Server. The response message indicates whether the request is succeeded or failed. Second, when the server allowed this connection from a OSPF API-Client, as a reaction it will open a reverse connection channel for asynchronous messages*. The asynchronous channel is used for sending the notifications from a client and/or a server. These notifications are one way messages and can point out new, updated, or deleted LSA(s). The Quagga OSPF API-client library provides *"ospf_ apiclient_ connect (char *host, int syncport)"* function for a client application to start a connection to the OSPF-API-Server.

- **Link-state database synchronization:** When the connection between OSPF API-Client and OSPF API-Server has been established, then the client application can send link-state database synchronization request to the server. In this case, the OSPF API-Server will send entire OSPF LSDB in a sequence of LSAs to the OSPF API-Client.

- **Opaque Type registration:** When the LSDB synchronization achieved, then the OSPF API-Client can register the opaque type which it plans to originate as an opaque LSA.

- **Origination of own opaque LSAs:** when the OSPF demon learnt that an opaque-capable neighbor's state is full the OSPF API-Server will notify the OSPF API-Client that it is ready to flood opaque LSAs. The OSPF API-Client application then can originates its own opaque LSA and invoke OSPF demon to flood it throughout the OSPF network.

- **Update own opaque LSA:** The OSPF API-Client can update the content of self-originated LSA and asks OSPF API-Server to re-flood it with a new content.

- **LSA update from neighbors:** Whenever any kind of LSAs picked up by OSPF demon, the OSPF API-Server will send a copy to OSPF API-Client

*Based on current implementation of the Quagga OSPF API the $asynchronousport = synchronousport + 1$

to keep it synchronized with Quagga router's LSDB.

- **Deletion of own opaque LSA:** The OSPF API-Client application can delete its opaque LSA from all OSPF routers by sending the deletion request to OSPF API-Server.

- **Connection shout down:** The OSPF API-Client can terminate the connection to OSPF API-Server by sending the shout down request. The client should delete all self-originated opaque LSA before sending a shout down request.

As discussed earlier in this section, the OSPF API-Server communicate with the OSPF API-Client application with the sort of notifications. Different OSPF API-Server's notification message types are shown in Table 2.7. It is worth highlighting that the OSPF API-Client application is responsible to handle the notifications from the server side. To achieve this goal, the client application should register its call back functions for each type of notifications. The OSPF API library provides the *"ospf_ apiclient_ register_ callback()"* function to register the client call back functions.

Table 2.7: OSPF API-Server notification message types.

Notification	**Description**
MSG_ READY_ NOTIFY	The OSPF demon is ready to generate the Opaque LSA.
MSG_ LSA_ UPDATE_ NOTIFY	The link-state update revived.
MSG_ LSA_ DELETE_ NOTIFY	The LSA is deleted from LSDB.
MSG_ NEW_ IF	New network interface is added to the network.
MSG_ DEL_ IF	An interface is deleted from the network.
MSG_ ISM_ CHANGE	The Interface state machine is changed.
MSG_ NSM_ CHANGE	The neighbor state machine is changed.

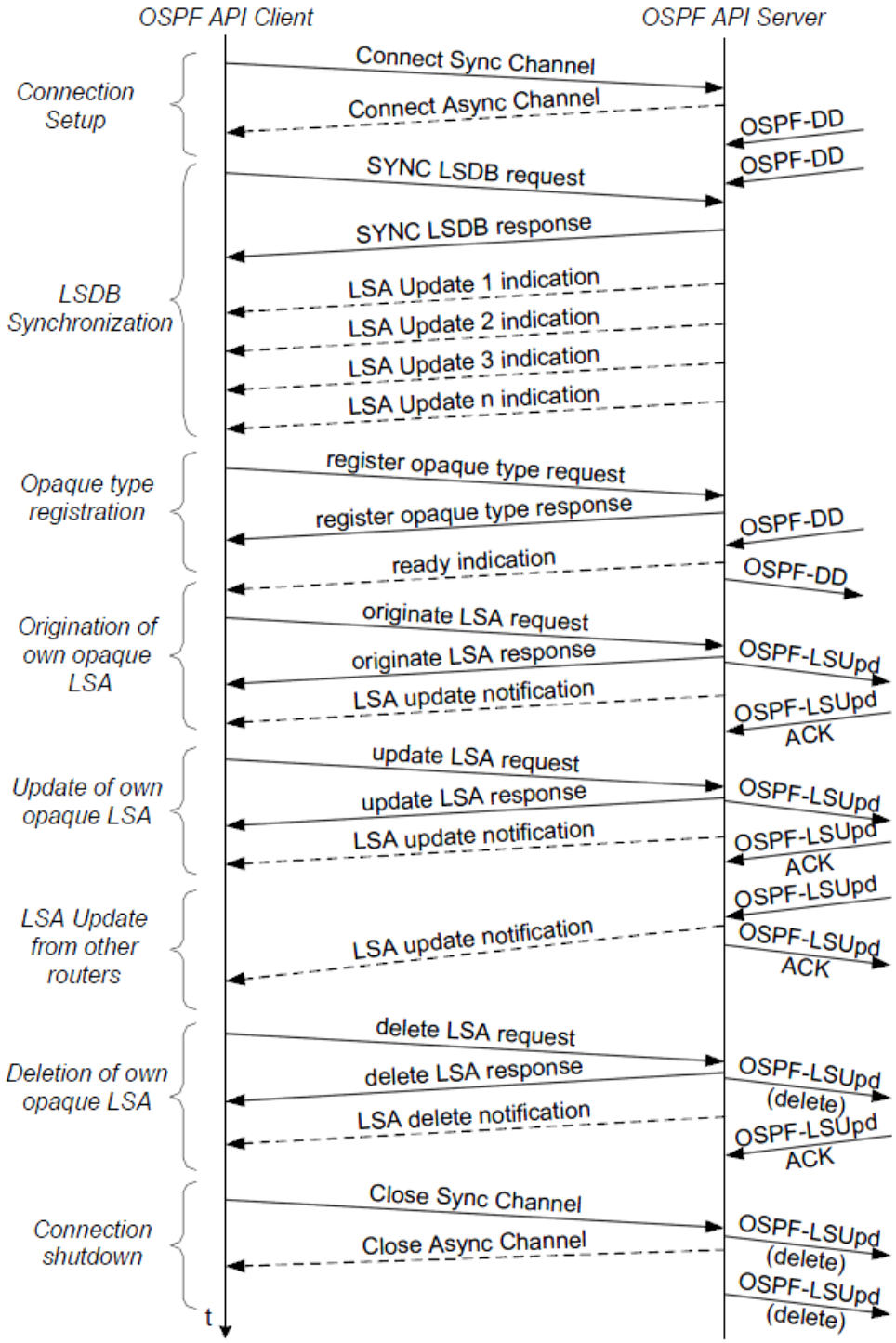

Figure 2.7: The OSPF API protocol states [61].

2.7 Related works

The work presented in this thesis project focuses on three research areas: monitoring of cloud resources, using opaque LSA for sending arbitrary information, and link-state update policy.

A number of authors have proposed a distributed cloud resource allocation system, see [21, 62, 63, 64]. These studies show that a fundamental part of resource allocation in a distributed cloud is monitoring of distributed resources. In fact, a cloud manager and resource allocation system needs to have up to date information about resource availability in each of the data centers in order to be able to make an appropriate decision. In a recent master's thesis, V. Visockas has shown how to visualize the interactions between services running in a data center and between data centers based on the information that can be obtained by agents running in the virtual machines or by information that can be observed by the routers [65].

Resource monitoring in a cloud has been subject of many works, see [7, 66, 67]. In these works, the authors mostly consider the monitoring techniques inside a data center and *not* monitoring the distributed data centers. This work dose not consider the methods of resource monitoring inside the data center. We assume the information about data center's functional capacity is available. Based on this assumption, we proposed a solution for monitoring distributed data centers *plus* network and link condition in a network embedded cloud.

Taking advantage of OSPF routing protocol's opaque LSA by proposing a new TLV to carry arbitrary information was subject of many works. R. Keller and B. Plattner extent OSPF protocol by proposing new opaque LSA TLV to carry a router processing attributes for their active network control software [68]. As another example, L.G. Zuliani and R. Pasquini proposed a new TLV to the OSPF-TE protocol to enable the Generalized Multi-Protocol Label Switching (GMPLS) to have most effective decision [69]. The OSPF-TE and all of its extensions (which use opaque LSAs) can be considered as a related work in this area. The OSPF-TE widely discussed in literature and books. In this work, the OSPF-TE was explained in section 2.3.4.

Link-state update policy has been a subject of much research, especially in QoS routing, see [53, 54, 55, 70, 71, 72]. These works mostly consider different link-state update policies for distributing available link bandwidth information to provide required information for QoS routing and TE. In a recent master's thesis, M. Noordermeer has shown that determining the appropriate parameters for the

link-state update policy is more influential on the amount of traffic and decision accuracy than the choice of the policy itself [73]. Link-state update policy and some results of mentioned related work are discussed in section 2.5 and chapter 5 when it was required.

To the author knowledge, this is the first work considering resource monitoring in a network embedded cloud. However, the works discussed above was used as a part of proposed solution in this work or considered the resource monitoring and resource allocation in general.

Chapter 3

Design

This chapter specifies the architecture and the design of a proposed solution. The information required to provide an extension to OSPF-TE to convey data center's resources is discussed. Finally, the proposed Cloud LSA architecture is presented.

3.1 Solution architecture

In this thesis project we use the OSPF protocol to transmit information about cloud resources and to provide the required information for a CMS. To achieve this goal, an extension to OSPF-TE is proposed as shown in Figure 3.1.

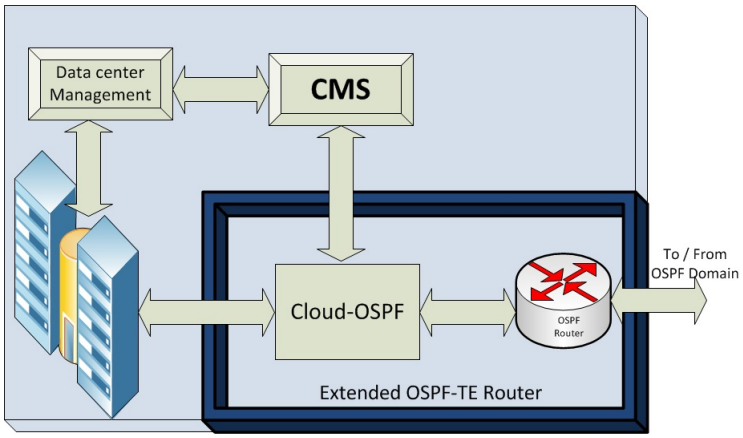

Figure 3.1: Proposed solution architecture.

The router extension module, called ***"Cloud-OSPF"***, is responsible for collecting resource information from (an embedded) data center. This information is distributed to the OSPF domain as Cloud LSAs via the OSPF router. The Cloud-OSPF module also is responsible for retrieving and analyzing received Cloud LSAs that were generated by other instance of the Cloud-OSPF module within the OSPF area. This information can be used to update the CMS. The CMS then can run any constrained-base algorithm to find the best instance(s) of the requested resources in the cloud network. Finally, the CMS tells the data center's management system to allocate resources to a user. In the following section, more details about the Cloud-OSPF module's design are explained.

3.2 Cloud-OSPF module's design

In this section, the design issues for the Cloud-OSPF protocol and router module are explored, followed by a description of the different parts of Cloud-OSPF protocol and router module.

3.2.1 Design issues

Two issues should be considered while designing an extension to OSPF-TE. The first issue is, choosing an OSPF router implementation. The second issue is, selecting a good model for an embedded data center.

3.2.1.1 Quagga as routing suite

As described in section 2.6, several OSR software implementations are available. The project goal is to add an extension to the OSPF-TE protocol in such a way that the extended OSPF-TE conveys information about a data center's resources in a new type of LSA. This goal determines the suitability of the proposed solution from both performance and robustness perspectives. However, the choice of router does not effect the project except the for the details of design and implementation. Quagga was chosen as a platform because of its active development community. Additionally, Quagga's OSPF API enables us to develop an extension to the OSPF-TE protocol *without* requiring extensive knowledge or understanding of the Quagga source code.

3.2.1.2 Embedded data center

The resource utilization modeling of an embedded data center directly affects the evaluation part of this project. This model should simulate the activity of a data

center. Unfortunately, neither cloud providers nor the literature in the field provide detailed information about how a data center's resources change during a business day. As a result, finding a realistic model of how cloud resources change is very important.

The author attempted to get information about the energy consumption of a data center. The assumption was that when the data center serves more requests that more resources are assigned and that during this time the data center will consume more energy. Based on this hypothesis we assume the following:

During the first hours of the day, the cloud, and as a consequence the data center, experiences a minimum number of resource allocation requests, thus the data center resource utilization has a minimum value. In the early morning hours as people start to work, the cloud experiences a higher rate of resource allocation requests which leads to a higher utilization of the data center's resources. The utilization will reach its highest level around midday. Finally, in the evening, users will send resource de-allocation requests which decrease the resource utilization. Finally, during late night, the utilization returns to the lowest level. This argument is applicable to a data center where most of users are located in a single geographical area and assumes that the demands are largely related to human users making requests, as opposed to batch processing.

3.2.2 Solution Design

Cloud-OSPF is an extension to the OSPF protocol to extend it in such a way that it transmits virtualization and processing-related details of a network embedded cloud. The implementation of Cloud-OSPF is done by implementing two separate modules (as shown in Figure 3.2): *"Cloud-OSPF-Receiver"* and *"Cloud-OSPF-Sender"*. Utilizing two separate modules to implement the Cloud-OSPF protocol offers flexibility, as either the Cloud-OSPF-Receiver or Cloud-OSPF-Sender can be shut down independently. Shutting down one or the other will reduce the router's processing overhead.

3.2.2.1 Cloud-OSPF-Sender

The **"Cloud-OSPF-Sender"** is responsible for reading and examining the *"Cloud Resource DB"*. The Cloud Resource DB contains information about the usable resources of an embedded data center (e.g. CPU, RAM, and storage). The Cloud-OSPF-Sender can utilize the OSPF router to inform other OSPF nodes by sending Cloud LSAs when significant changes occurs in this instance of the Cloud Resource DB. The Cloud-OSPF-Sender and Cloud-OSPF-Receiver

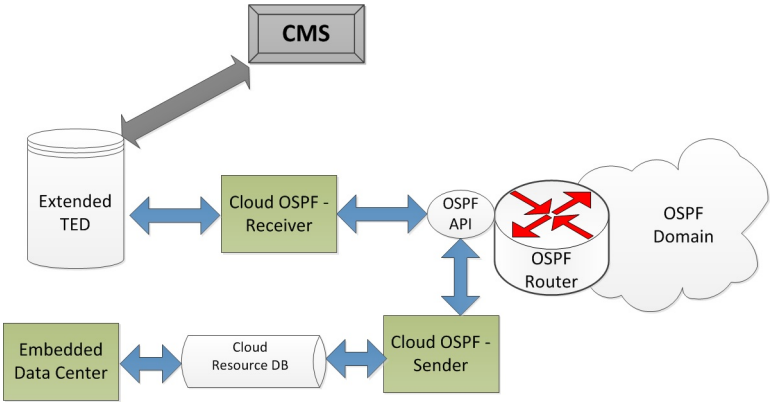

Figure 3.2: Design of the proposed solution

communicate with the Quagga router via the OSPF API. It is worth mentioning that the information about network capacity (e.g., network topology and link characteristics) will be provided by OSPF and OSPF-TE and *not* the Cloud-OSPF-Sender.

3.2.2.2 Cloud-OSPF-Receiver

The **Cloud-OSPF-Receiver** is responsible for keeping the *"Extended TED"* up to date. The Extended TED contains information about all the embedded data centers *plus* the network's topology and link attributes. To achieve this goal, the Quagga router forwards a copy of all LSAs to the Cloud-OSPF-Receiver. The Cloud-OSPF-Receiver analyzes and extracts information from these LSAs to update the Extended TED. The CMS of the network embedded cloud can query the Extended TED to obtain information required to make an appropriate decision.

3.3 Cloud LSA

As discussed in section 2.3.2, The OSPF protocol utilizes LSAs to describe the local network's status and to provide information about the network to other nodes. As discussed in section 2.3.3, the OSPF router uses an opaque LSA to support these generalized LSAs and floods this data within an OSPF area. In this section, the details of an extension of the OSPF-TE protocol are given. A new LSA, named ***"Cloud LSA"***, is proposed to convey information about embedded data center resources to the CMS. This Cloud LSA is as an enhancement to OSPF-TE protocol as it makes use of the TE LSAs, which are opaque LSAs with an area wide flooding scope (LS type 10).

3.3.1 Cloud LSA format

The Cloud LSA starts with the standard TE LSA header with LS Type 10 and opaque Type 1. The payload portion of the Cloud LSA contains information about a data center's resources. There are two ways that we could store the information about the data center resources in the payload. the first alternative is to propose a new type of TE TLV, and another option is to extend the existing TE TLV by adding one or more sub-TLVs.

Additionally, there are different ways to described the data center's resources. One approach is to place all of the information about available resources in the value part of a top-TLV (or a sub-TLV). In this case, this top-TLV (or sub-TLV) can supply all the required information (such as CPU, RAM, storage, and/or data center's location). The second approach is to introduce different sub-TLVs (or sub-sub-TLVs) for each resource. In this second approach, one sub-TLV (or sub-sub-TLV) is required for each data center resource. Both options have benefits and weaknesses. Selecting the best alternative requires more research and analysis which are out of scope of this work. Here we will utilize the first approach, thus we place all of the available information about the data center's resources in the value portion of a top-TLV (or sub-TLV).

The registered top-TLV for a TE LSA was described in Table 2.4 on page 28. We propose a new top TLV with a value 6 and with the name ***"Cloud TLV"***. This Cloud TLV can hold a data center's resource information in its value portion in two ways. The first approach is to place the data entirely in the value portion of the Cloud TLV as shown in Figure 3.3, and another approach stores the data center's information in different sub-TLVs of the Cloud TLV (such as a CPU-sub-TLV, RAM-sub-TLV, and Storage-sub-TLV) as shown in Figure 3.4. In this case, the Cloud TLV can be composed of any combination of sub-TLVs.

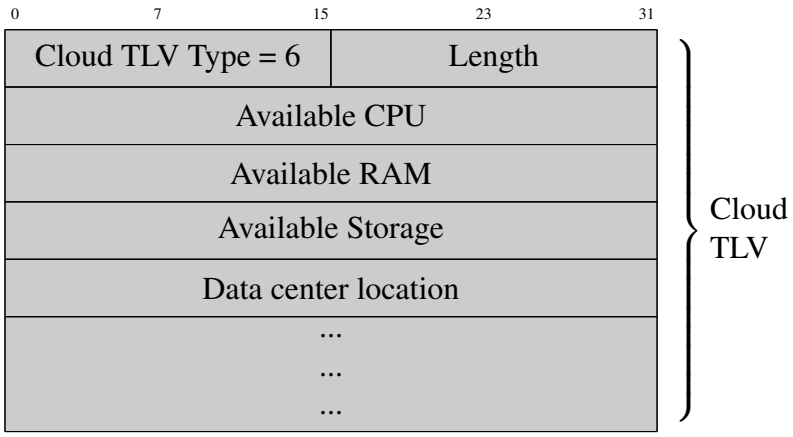

Figure 3.3: Cloud TLV - storing all the required data about an embedded data center entirely in a value portion.

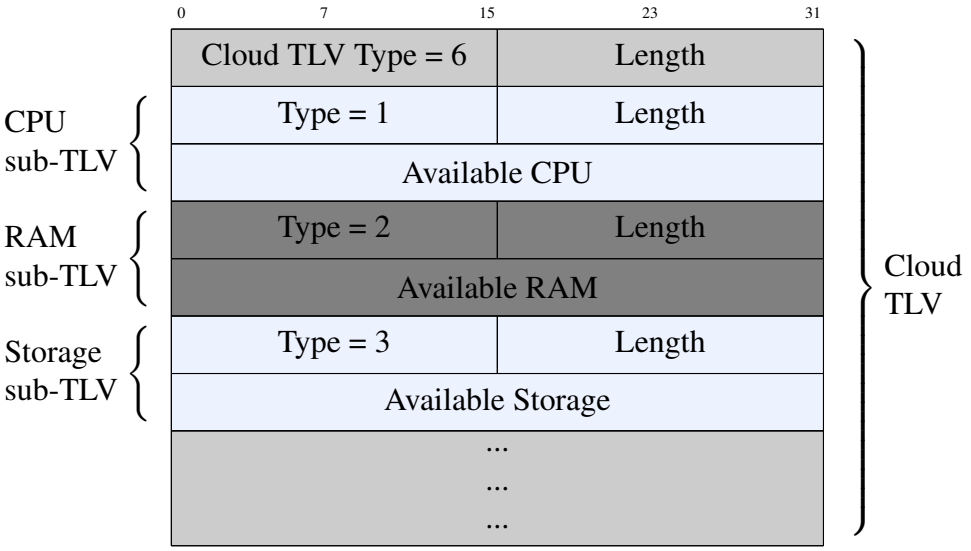

Figure 3.4: Cloud TLV - storing the embedded data center's information in different sub-TLVs.

It worth mentioning that the embedded data center can be realized using the compute capability of a router. The Node attribute TLV, which has type 5, carries the attributes associated with a router. Hence, the Node attribute TLV could accommodate information about the compute capability of a data center in the form of a new sub-TLV. Considering the registered sub-TLV for Node attributes TLV shown in Table 2.5 on page 29, a new sub-TLV for a node attribute TLV is proposed with the name of ***"Cloud sub-TLV"*** and has a type value equal to 3 (as

shown in Figure 3.5). The Cloud sub-TLV can hold the embedded data center's resources information.

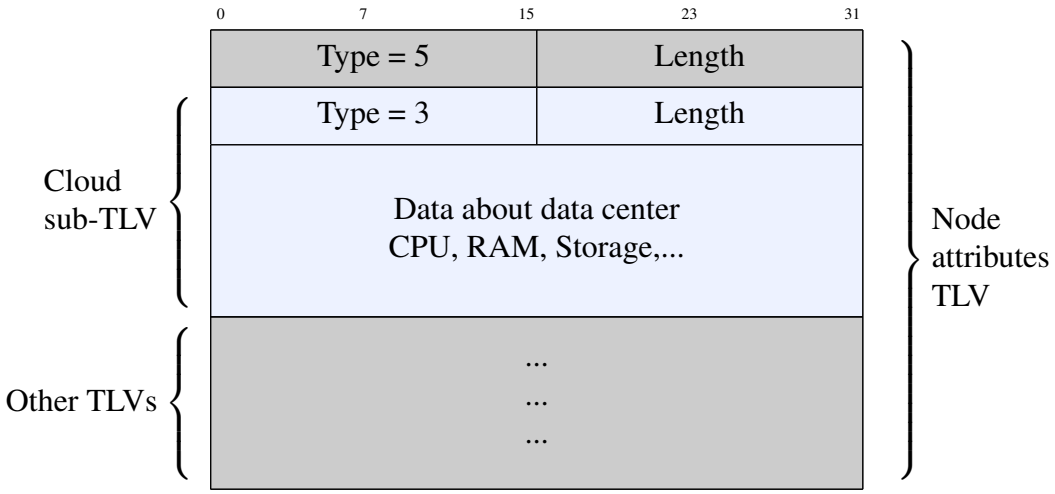

Figure 3.5: A Cloud sub-TLV for a Node attribute TLV

Each of the options to place data center resources in a TE LSA, the Cloud TLV and Cloud sub-TLV for Node attributes TLV, has advantages and disadvantages. Selecting the best way to encode the data center resources information requires more research and analysis which are out of scope of this work. Here we will assume that the data center resource are encoded in the Cloud sub-TLV of the Node attributes TLV of a TE LSA. One reason for making this choice is that it is easy to add new sub-TLVs if there are additional resources that we want to describe, for example we could add a sub-TLV to indicate what type of processor is available.

When it comes to the CMS, a data center can be described in terms of the available CPU, RAM, and storage resources plus the data center's location. Another consideration is which parts of the data center's resource information is required by the CMS. Placing more information in the Cloud LSA provides the CMS with more information, which might help the CMS to make a better decision. On the other hand, adding more information increases the network overhead of each Cloud LSA. As each OSPF router in the area must process each LSA, this can have a negative effect on each of these OSPF router's performance. We will assume that the Cloud LSA contains four parameters: available CPU in units of clock frequency in Gigahertz (GHz), available RAM in Gigabytes (GB), available storage in Gigabytes (GB), and the location of the data center.

The sizes of the Cloud TLV value portions for the CPU, RAM, and location are an open question in this step. As discussed earlier, these portions represent the data center's available resources and the data center's location. Finding the optimal size for data center's resource (i.e., CPU, RAM, and storage) requires more information about telecommunication operators data center's capacity, which is not clear yet. The value portion for data center's location could be an index number or the physical location of the data center if World Geodetic System 1984 (WGS84) coordinates. More research and study needed to find an optimal size for Cloud LSA value portions which was out of scope of this work and left for future work. In this project, we make the value parts of Cloud TLV for cloud resources as fixed unsigned 32 bit fields which can represent at max *2,147,483,647* (i.e., $2^{(32)} - 1$). Additionally, the value portion of data center's location is considered as fixed 32 bits string.

3.3.2 Summery

The proposed Cloud LSA starts with a standard TE LSA header with LS type 10 and Opaque Type 1. We propose a Cloud sub-TLV for the Node attribute TLV. The Node attribute TLV has type 5 and its Cloud sub-TLV has a type 3. This Cloud sub-TLV contains information about data center's resource (specifically available CPU, RAM, storage and the data center's location). The proposed Cloud LSA format is shown in Figure 3.6. It is worth highlighting that the proposed Cloud LSA contains 44 bytes of data.

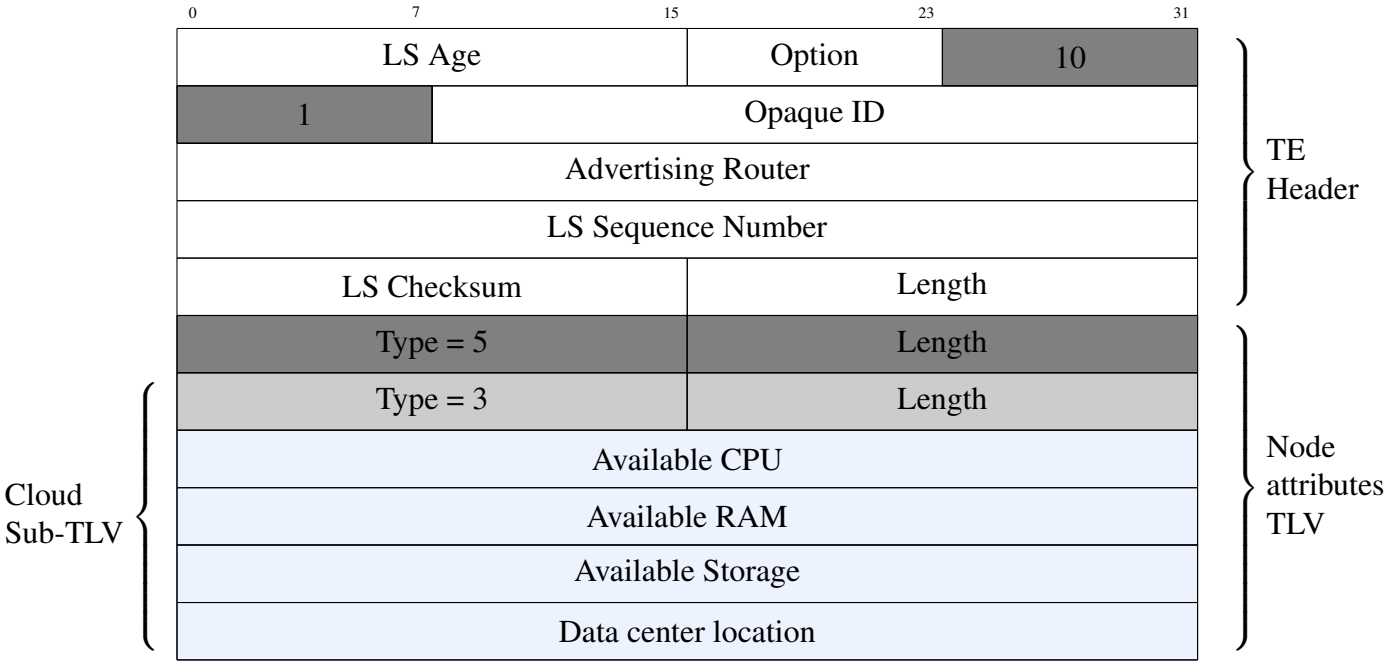

Figure 3.6: Cloud LSA format.

Chapter 4

Implementation

This chapter explains implementation of the proposed solution design presented in the previous chapter, including data center utilization simulator, Cloud-OSPF-sender, and Cloud-OSPF-Receiver. Some additional concepts are discussed when these concepts are needed to enable the reader to understand the implementation of the proposed solution.

4.1 Data center resource utilization module

The embedded data center module simulates a data center with varying levels of activity. The embedded data center module is written in the C programming language. This module simulates the data center's activity over 24 hours as described in section 3.2.1.2. This module is only used for testing and a real-world system would replace this module by code that would get the resource information from a data center management system (e.g., Amazon's CloudWatch [74], CloudClimate [75], and CloudKick [76]).

4.1.1 Data center module's flowchart

This module operates as shown in Figure 4.1. The embedded data center simulation module behaves as a data center with some pre-defined capacity. We assume that the data center's capacity can be described by specifying the CPU's clock rate in GHz, the amount of RAM in GB, and the amount of storage in GB. The data center simulator serves user requests by allocating or releasing data center resources. The module simulates user requests based on a bounded request arrival rate. The request arrival rate describes how often this data center receives another request. This request arrival rate is configurable in our model.

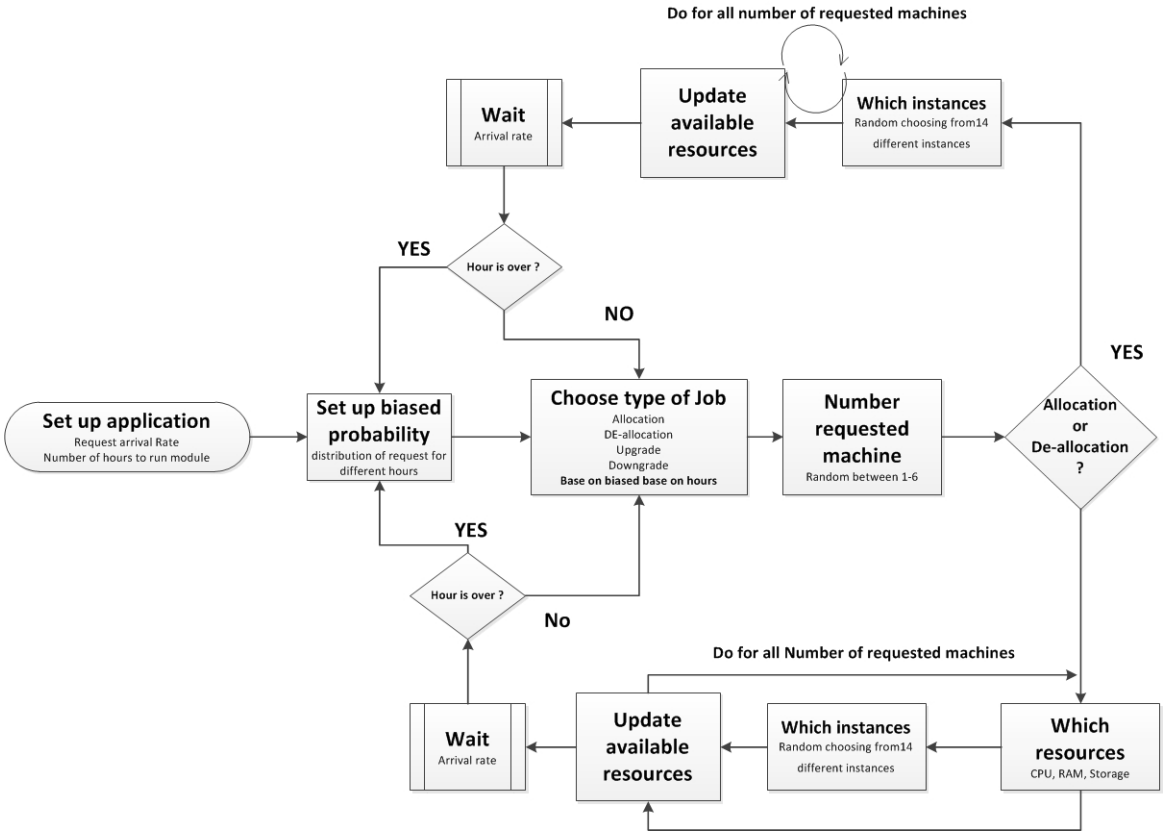

Figure 4.1: Data center resource utilization modeling flowchart. This chart skipped some features of application (such as connecting to sqlite database details, biased Probability distribution mechanism, and data center's monitoring system).

A user request is described by three parameters: *"job type"*, *"instance type"*, and *"number of instances*. The embedded data center module supports four pre-defined job types: *"Resource Allocation"*, *"Resource Upgrade*, *"Resource De-allocation"*, and *"Resource Downgrade"* as shown in Table 4.1. It is worth mentioning that the Resource Upgrading and Resource Downgrading can apply to CPU, RAM, storage, or any combination of them.

Table 4.1: Data center job types.

Type	Job Name	Description
1	Resource Allocation	Request for allocating a virtual resource. User needs to send this request before sending a Resource de-allocation, upgrade, or downgrade.
2	Resource De-allocation	Request to release an allocated virtual resource.
3	Resource Upgrade	When the user already has some resources they can ask the data center to increase the allocation of these resources.
4	Resource Downgrade	When the user already has some resources they can ask the data center to de-allocate some of these resources.

The "instance type" is one of 14 pre-defined instances. Each instance is described by some amount of CPU, RAM, and storage. The number of instances is a random integer value between one and six. The user request parameters will be randomly selected from possible sets of the above options. More information about the data center capacity, type of instances, and part of embedded data center simulation module's source code are presented in Appendix A.

4.1.2 Data center resource utilization results

The embedded Data Center module was executed 100 times to have results with a confidence interval of 95%. Each round of run simulated 24 hours and the data center module started at midnight, i.e., 00:00. This enables us to observe the behavior of a simulated data center over one business day.

As in simulated center module the user request types are biased based the day's hours (e.g., in working hours the probability of resource request and resource

upgrade is higher than the none working hours), the data center utilization changes frequently and it has potentially curve point in each hour. The mean (i.e., opposed to the median values) for utilization in each hour enables us to have a general overview on data center utilization behavior. The *mean* of *average CPU utilization* and its upper and lower bounds of a 95% confidence interval (CI) for a simulated data center over 24 hours is shown in Figure 4.2. The first hour in graph depicts the mean of average CPU utilization from 12:00 A.M till 12:59 A.M and 24th hour in graph depicts the mean of average CPU utilization from 11:00 P.M till 11:59 P.M. The red lines in graph present each hour's upper and lower bounds of a 95% CI for mean.

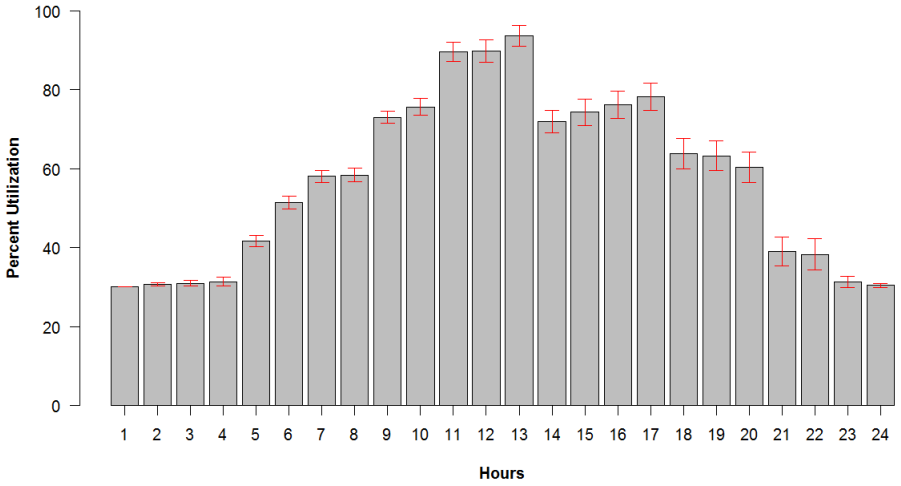

Figure 4.2: Data center's *mean* of *average CPU utilization* per hours with upper and lower bounds of a 95% CI.

The *mean* of usable CPU capacity of the data center and its upper and lower bounds of a 95% CI per second over 24 hours is shown in Figure 4.3. The corresponding graphs of the data center's *mean* of average utilization and *mean* of usable capacity for RAM and storage are given in Appendix B.

The data center resource utilization module logs the data center's available resources in the Resource Database as was shown in Figure 3.2 on page 44. Third party software can use this database to learn about the current status of the data center. The Resource Database is built upon an SQLite [77] database and contains information about simulated data center's resources with records such as that shown in Table 4.2. Additionally, this module includes a live monitoring system

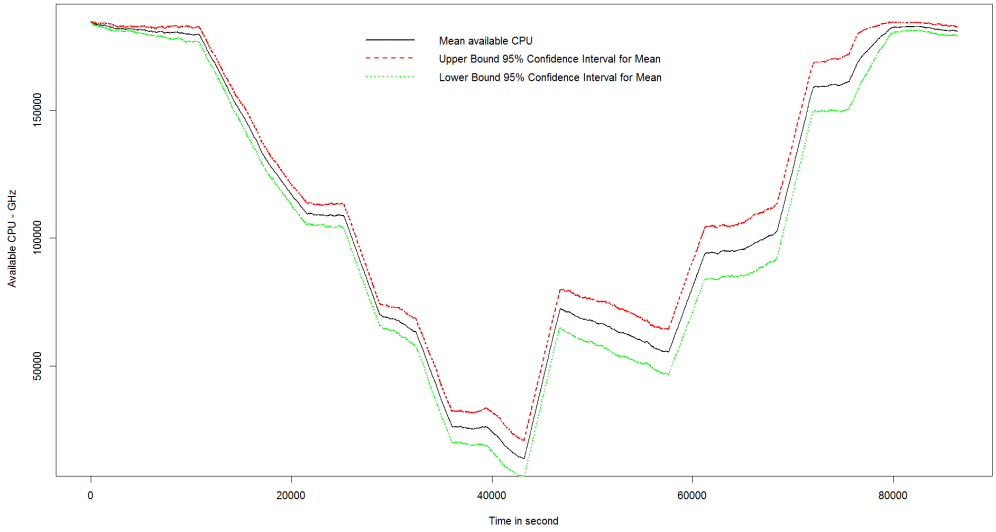

Figure 4.3: Simulated data center's *mean* available CPU capacity (GHz) per second and its upper and lower bounds of a 95% CI.

to enable a user to monitor available capacity and utilization of CPU, RAM and storage within a simulated data center.

Table 4.2: Resource Database format

CPU	**RAM**	**Storage**	**Location**
::INTEGER	*::INTEGER*	*::INTEGER*	*::TEXT*

4.2 Cloud-OSPF-Sender module

The Cloud-OSPF-Sender is responsible for several tasks, including monitoring the data center's available resources, checking the data center's resources based on an update policy, and sending information about the data center's available resources in the form of opaque LSAs into the OSPF domain, as explained in section 3.2.2.

The update policy is the most crucial part of the Cloud-OSPF-Sender as this policy will determine how the Cloud-OSPF-Sender behaves when it detects changes in the data center's resources. The update policy determines whether the change in the data center's usable capacity is significant or not. The choice of update policy directly affects the performance of the proposed solution. As a

result, it is beneficial to select a good policy in order to avoid unnecessary updates. The Cloud-OSPF-Sender uses *relative threshold update policy*. More information corresponding selecting an update policy is given in chapter 5.

The Cloud-OSPF-Sender module is written in the C programming language. It uses threads to provide concurrency. As an example, one thread can be run for reading the data center's resources while another thread is responsible for packing a Cloud LSA and sending it to the OSPF network. Also, the monitoring system for the embedded data center is run in parallel with other threads. In addition, in the future, we can add more threads (with different responsibilities) to this module without effecting the functionality of other parts of it.

The Cloud-OSPF-Sender is compatible with the Quagga router and employs the Quagga OSPF-API for low-level communications, see section 2.6.1. The flowchart for this application is shown in Figure 4.4. The Cloud-OSPF-Sender application allows the Cloud-OSPF system operator to configure several parameters before execution such as:

Target router IP address	defines the administrative IP address of the target router. The application uses this IP address to connect to Quagga OPSF API.
Port for synchronous connection	define the port number that Cloud-OSPF-Sender module uses for synchronous connection to OSPF API-server.
Opaque ID and Opaque Type	define the Opaque ID and Opaque type for the Cloud LSA. The Opaque ID can be any arbitrary number. Based on the discussion of Cloud LSA in section 3.3, the Opaque Type should be 10 (TE LSA), the top-TLV type is 5 (Node attributes TLV), and the sub-TLV type is 3 (Cloud sub-TLV).
Threshold value	the module uses the threshold-based with relative change update policy (See Chapter 5). This parameter defines the threshold value as a percentage change of the resource.
Interval value	defines the interval between when the module checks the data center's Resource DB. The default value for this parameter is MinLSInterval (i.e., 5 seconds).

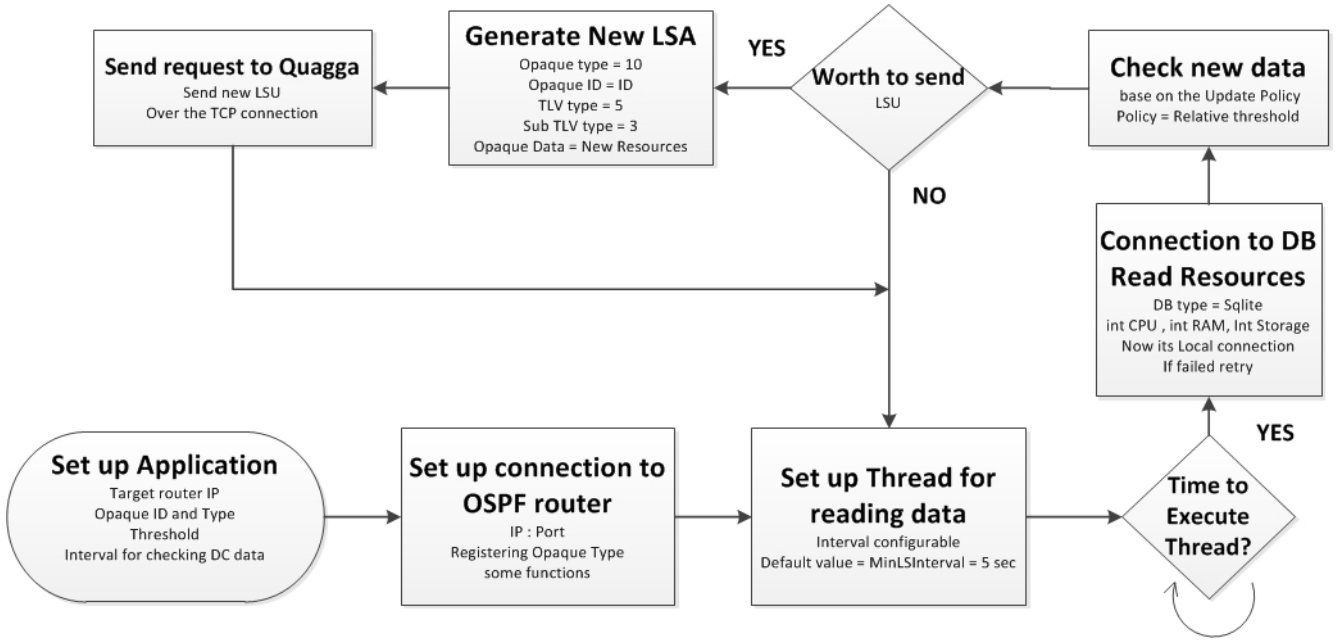

Figure 4.4: Cloud-OSPF-Sender module flowchart. This chart skip details for some parts of module (e.g., steps of connection to Quagga router, steps of Connection to data center's resource DB, and detail about update policy).

When the Cloud-OSPF-Sender starts, it will create a TCP connection to the Target router's IP address using "NEC-OSPF-TE*" port number. The value of NEC-OSPF-TE is configurable in our model. Next, it will register the given Opaque Type and ID to OSPF API-Server. Cloud-OSPF-Sender will also register its own call back function to OSPF API-Client libraries. The call back functions are responsible for handling the OSPF API-Server notification messages. The OSPF API protocol was discussed in section 2.6.2.

When the client-server connection established, the Cloud-OSPF-Sender creates a thread for reading the data from the data center's Resource DB. This thread is periodically invoked based on a timer which is set to the Interval time. The Cloud-OSPF-Sender module will re-create this thread and check the current status of data center's Resource DB when the timer expires. If the application finds *significant* changes up on update policy for any resource, it will initialize a new thread to generate a new Cloud LSA. This thread will ask the Quagga router to send the Cloud LSA in the form of a LSU into the OSPF domain. The generated Cloud LSA contains the latest information about the data center's resources. The Cloud LSA format was discussed in section 3.3. More information including the Cloud-OSPF-Sender's source code is given in Appendix C.

4.3 Cloud-OSPF-Receiver module

The Cloud-OSPF-Receiver module is written in the Python programming language. This application is compatible with the Quagga router and uses the Quagga OSPF-API for low-level communications, as described in section 2.6.1. The Cloud-OSPF-Receiver synchronizes itself with the Quagga TED. Therefore, the Quagga router forwards a copy of all LSAs received from the OSPF domain to Cloud-OSPF-Receiver. The flowchart of this module is shown in Figure 4.5.

*This is a port number that should register by IANA for a network embedded cloud monitoring system

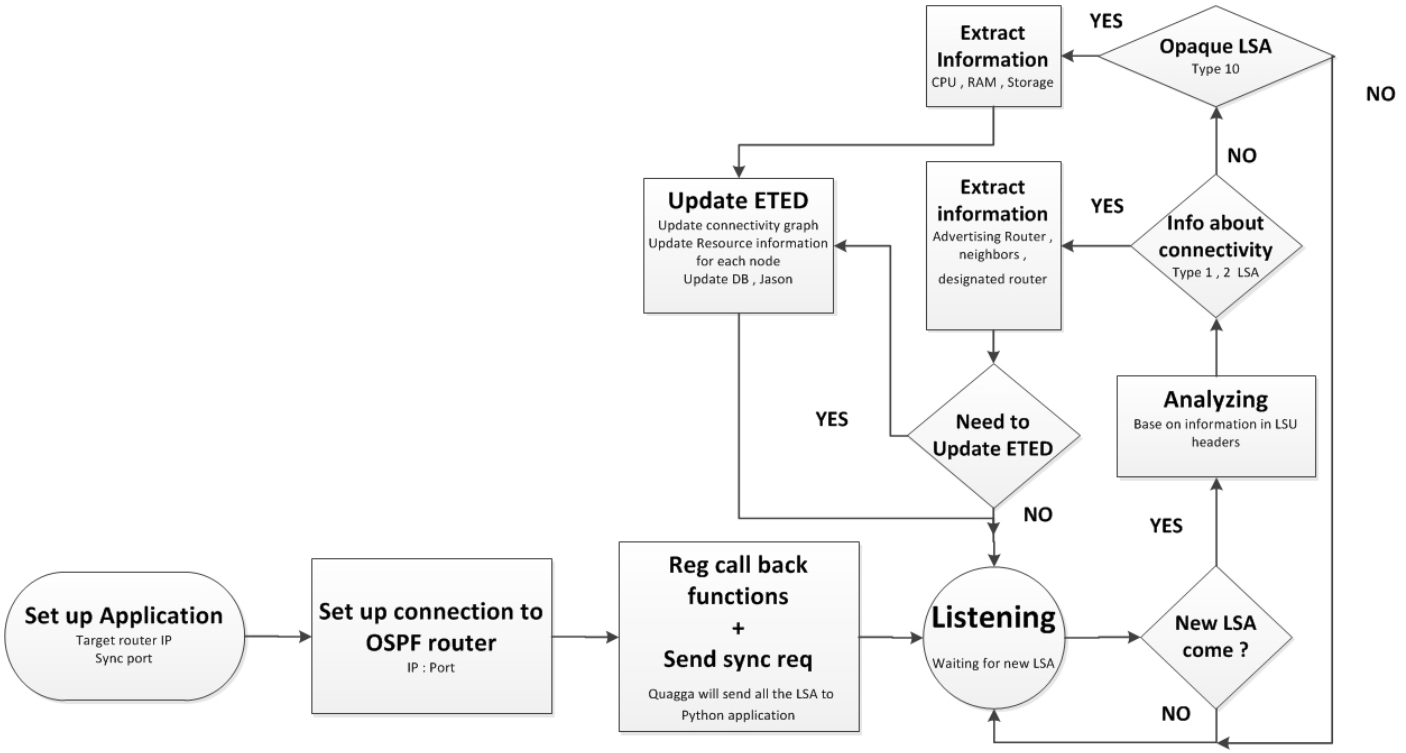

Figure 4.5: Cloud-OSPF-Receiver module flowchart. This chart skip details for some parts of module (e.g., steps and details of : connection to Quagga router, registering call back functions, extracting and analyzing the received LSAs, and connection and updating the Extended TED, and detail about update policy).

This module allows the user to configure a target router's IP address and required port number. These parameters define the administrative IP address of the target router and the port number which is used for synchronous connection to OSPF API-server. The Cloud-OSPF-Receiver uses these IP address and port to initial a connection to the Quagga OPSF API.

When the Cloud-OSPF-Receiver starts it creates a TCP connection to the target router's IP address using "NEC-OSPF-TE" port number. The value of NEC-OSPF-TE is configurable in this module. Next, Cloud-OSPF-Receiver will register its own call back function to OSPF API-Client libraries. The call back functions are responsible for handling the OSPF API-Server notification messages. The Cloud-OSPF-Receiver connection sends a link-state database synchronization request to the OSPF API-Server over the accomplished client-server connection. After the Quagga router received this request, it will forward a copy each LSA that is received from the OSPF area to the Cloud-OSPF-Receiver. The Cloud-OSPF-Receiver listens to OSP API-Server and waits for OSPF API-Server notifications. It will execute one of its callback function when new notification received from server side.

When the Cloud-OSPF-Receiver receives a LSA from the Quagga router, it extracts the information from this LSA. After analyzing this, the application updates the Extended TED as needed. The Extended TED is built upon an SQLite database and contains information about network connectivity, TE metrics (if applicable), and embedded data center's resources. The CMS or any third party software can query the Extended TED data to monitor the network's and data centers status.

Chapter 5

Cloud resources *and* updates policies

Cloud resources are dynamic in nature. As a result, having up to date information about cloud resources in a network embedded cloud requires frequent Cloud LSAs update. To enable good decisions by the CMS, it is essential to select an appropriate policy for when to send Cloud LSAs. Sending *very frequent* Cloud LSAs updates can result in high traffic which will negatively impact the performance of the OSPF network. On the other hand, sending the *infrequent* Cloud LSA updates may result in losing synchronization between the Extended TEDs of the OSPF routers in the network, and as a consequence the CMS may make **"bad"** decision. A bad decision can be characterizes as:

Over reservation In this case, the "bad" decision means that resource management system assumes a data center in the network embedded cloud has sufficient resources, according to the information in the Extended TED, when in fact it does not have sufficient resources.

Under reservation In this case, the "bad" decision means that resource management system assumes a data center in the network embedded cloud has insufficient resources, according to the information in the Extended TED, when it actually has sufficient resources.

As discussed in section 2.5, several update policy are available, such as immediate, periodic, threshold-based, and class-based update policy. This chapter provides analysis on all mentioned update policies for sending a cloud resource information and concludes with a discussion. To have these data we run a proposed data center model (see section 4.1 for 100 times to have analysis results with a confidence interval 95%. During the running period, we gathered the data center's available resource information with an interval of resource arrival rate (i.e 1 second in experiments). Next, different update policies are applied to the data

center's resource information.

5.1 Immediate update policy

The *immediate update policy* triggers an update whenever there is a change in available resources. This policy maximizes the accuracy of resource management but dramatically increases the network load.

As discussed in section 2.4, the OSPF routing protocol is rate limited in generating new LSAs by MinLSInterval timer. The default value of this timer is 5 seconds. Hence, this policy cannot generate a new Cloud LSA less than 5 second interval. As a result, when the data center's resources change more than one time in 5 seconds interval this policy acts like a periodic update policy with an interval value 5 seconds.

5.2 Periodic update policy

In a *periodic update policy*, the originating router periodically advertises LSA update after the hold-down timer expires. This policy does not actually consider the extent of available resources but simply provides updates about available resources. As a result, in this update policy the number of updates can be calculated beforehand. The relation between the hold-down timer value and the number of updates for a data center's resources in one day is shown in Figure 5.1. It is worth mentioning that based on MinLSInterval timer of OSPF routing protocol the minimum hold-down timer value for a periodic update policy is limited to 5 seconds.

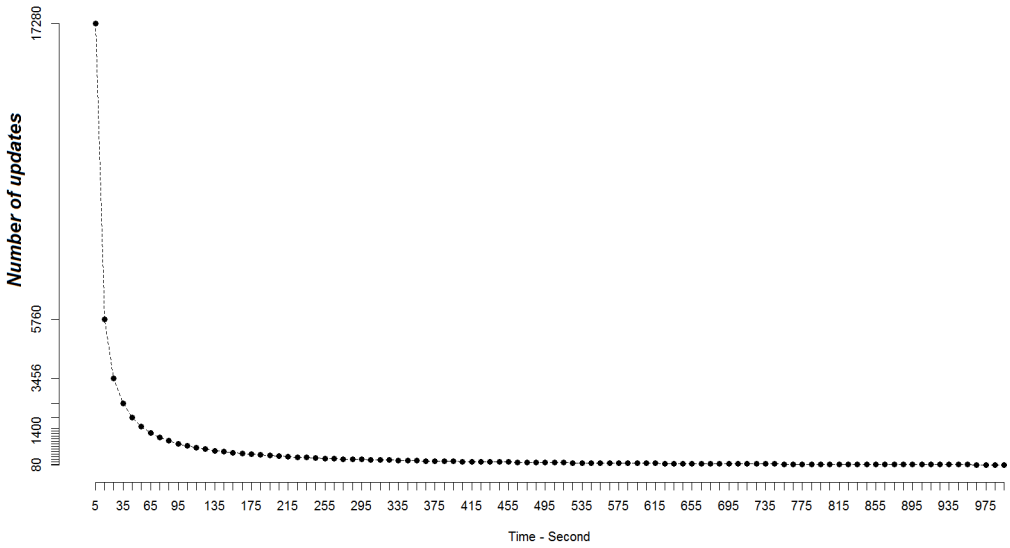

Figure 5.1: Number of updates per different hold-down timer values.

With a small value as a hold-down timer, the periodic update policy provides more information about data center's resources, but with a cost of sending more updates. As shown in Figure 5.1, the maximum number of updates in one day for a single data center is 17280 and accrues when a hold-down timer is set to 5 seconds. By increasing the hold-down timer value, the number of updates will decrease. However, the number of updates is not a sufficient factor to make a decision on hold-down timer value. It is also important to consider how the receiver node(s) (e.g., CMS) in an OSPF domain will views the data center's resources with different hold-down timer values. Figure 5.2 shows how CMS views a data center's available CPU capacity with different hold-down timer values 3600, 1000, and 200 seconds. The complete set of results for a data center's CPU, RAM, and storage capacity with the different hold-down timer values are given in Appendix D.

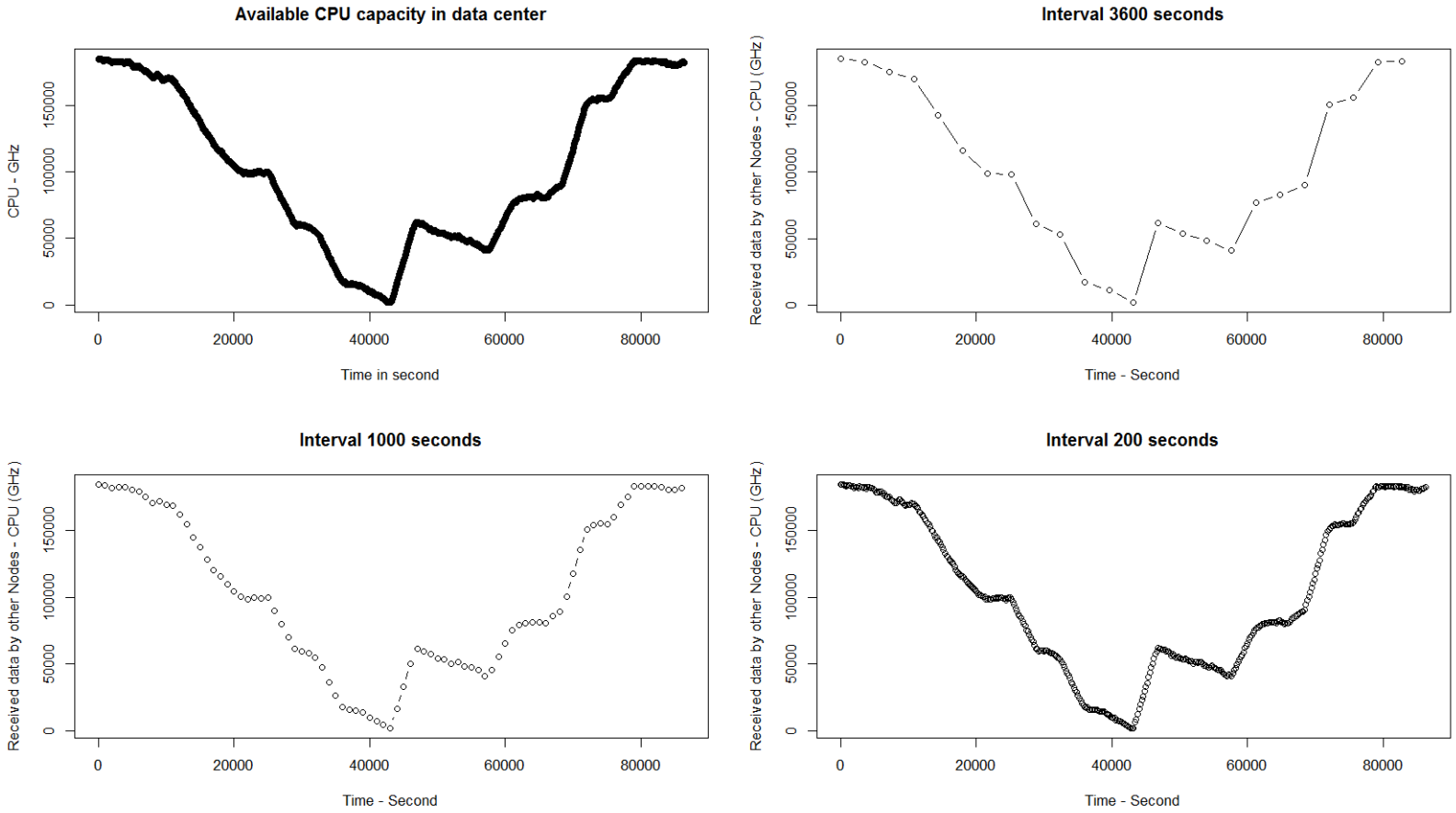

Figure 5.2: How a cloud management system views cloud resources with different hold-down timer values 3600, 1000, and 200 seconds, the updates points are plotted a circles in each graph.

Considering that the periodic update policy triggers a new update regardless how the data center's resource changes, a significant change can be ignored (for a period of time), and unnecessary LSA updates can occur often (especially if the period is short). Figure 5.3 visualized this problem for a hold-down timer value 700 seconds. In this Figure, the periods that are pointed with red are the periods that the periodic update send a new update while there are not significant changes, and the periods pointed with blue shows the periods that the periodic update policy ignores notable changes.

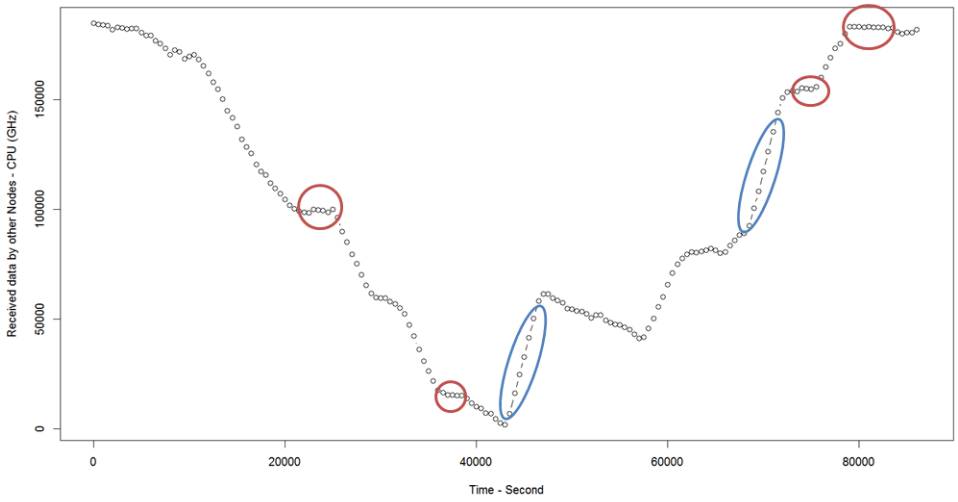

Figure 5.3: The periodic update policy can ignore necessary LSA updates (pointed in blue), and send unnecessary LSA updates (pointed in red), the updates points are plotted a circles.

5.3 Class-based update policy

As described by equation 2.4 on page 32, in a Class-based update policy the available capacity is divided to the number of classes. This policy triggers an update when the available capacity passed the class boundary. The classes size in this update policy determined by two key factors: a base factor "β" ($\beta < 1$), and a growth factor "f".

5.3.1 Equal-sized classes

When a growth fact "f" is equal to 1, the classes are equal-sized. In this case, the number of equal-sized classes (N_{eq}), can be calculated as shown in equation 5.1.

$$N_{eq} = \frac{1}{\beta} \tag{5.1}$$

where the "β" denotes a base class factor and determines each class size. In order to find an appropriate base factor, the *equal-sized class based update policy* with different base factor values was applied to the data center model which was described in section 4.1. Figure 5.4 shows how the base factor value effects on the number of updates due to simulated data center's CPU capacity changes.

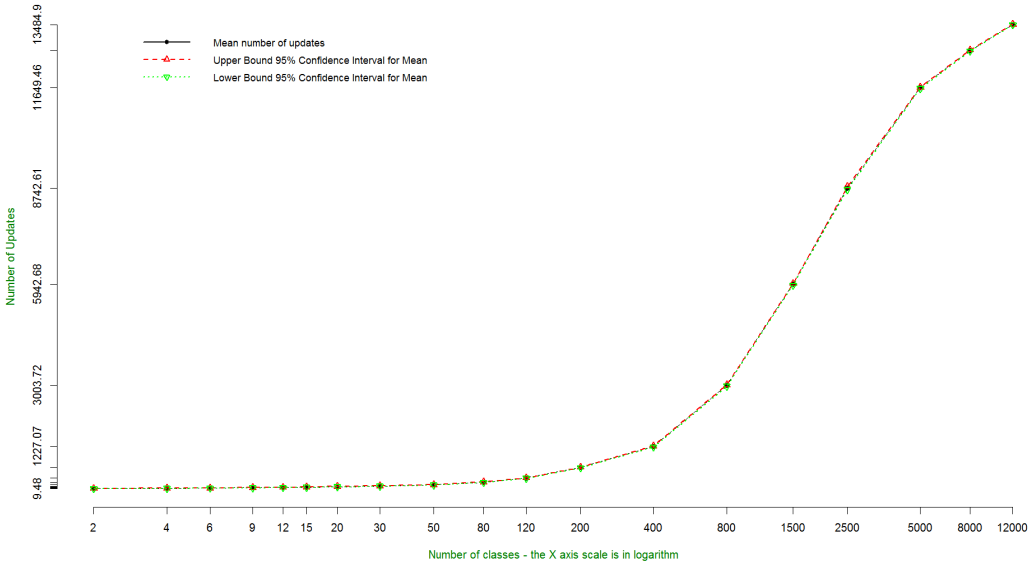

Figure 5.4: Number of updates per different number of classes (N_{eq}). The "X" axis are in logarithm scale.

This experiment clearly illustrates the hypothesis that the smaller base factor "β" value (higher N_{eq} value) results in sending fewer updates than dose a higher "β" value (smaller N_{eq} value). While reducing the number of Cloud LSA updates will reduce the network load, there is a trade-off between the update frequency and accuracy of cloud resource management system information about cloud resources.

As discussed earlier, in a class-based update policy a new update triggers when the available capacity passes a class boundary, As a result, it can generate lots of unnecessary updates when the available resources fluctuates around a class boundary. Figure 5.5 visualized this problem for a equal-sized class-based update policy. The periods that this update policy generates unnecessary updates are pointed with red. The hysteresis method can be used to prevent excessive unnecessary updates. In this method, a new update triggers when not only the available capacity passes the class boundary but also the magnitude of the changes are significant (e.g., the available capacity passes the middle of adjacent classes) [78].

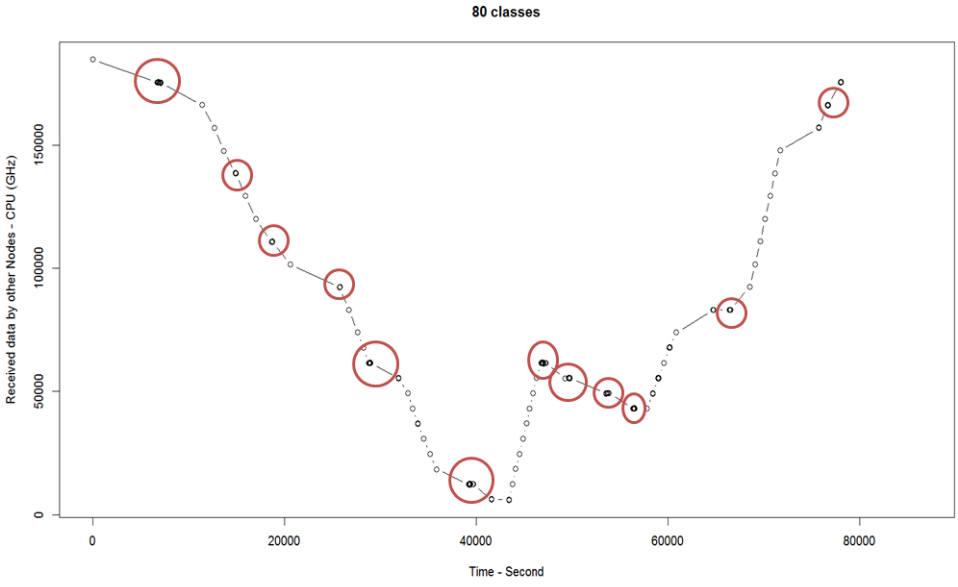

Figure 5.5: Equal-sized class-based update policy. How the update policy send updates due to changes in data center's available CPU capacity when number of classes $N_{eq} = 80$. The periods pointed in red show the duration that this policy generates unnecessary updates.

However, the number of updates is not a sufficient factor to make a decision on the number of classes in the equal-sized class-based update policy. It is also important to consider how the receiver node(s) (e.g., CMS) in an OSPF domain views the data center's resources with different number of classes. The analysis results enable us to see how the update policy reacts to changes in the data center's resources and how well updates can describe the data center's resources. The result for N_{eq} values 120, 200, and 400 are shown in Figure 5.6. The complete set of results are given in Appendix E. These Figures enable us to see the inaccuracy of the resource management system's view over the state of the cloud resources in some period of the time.

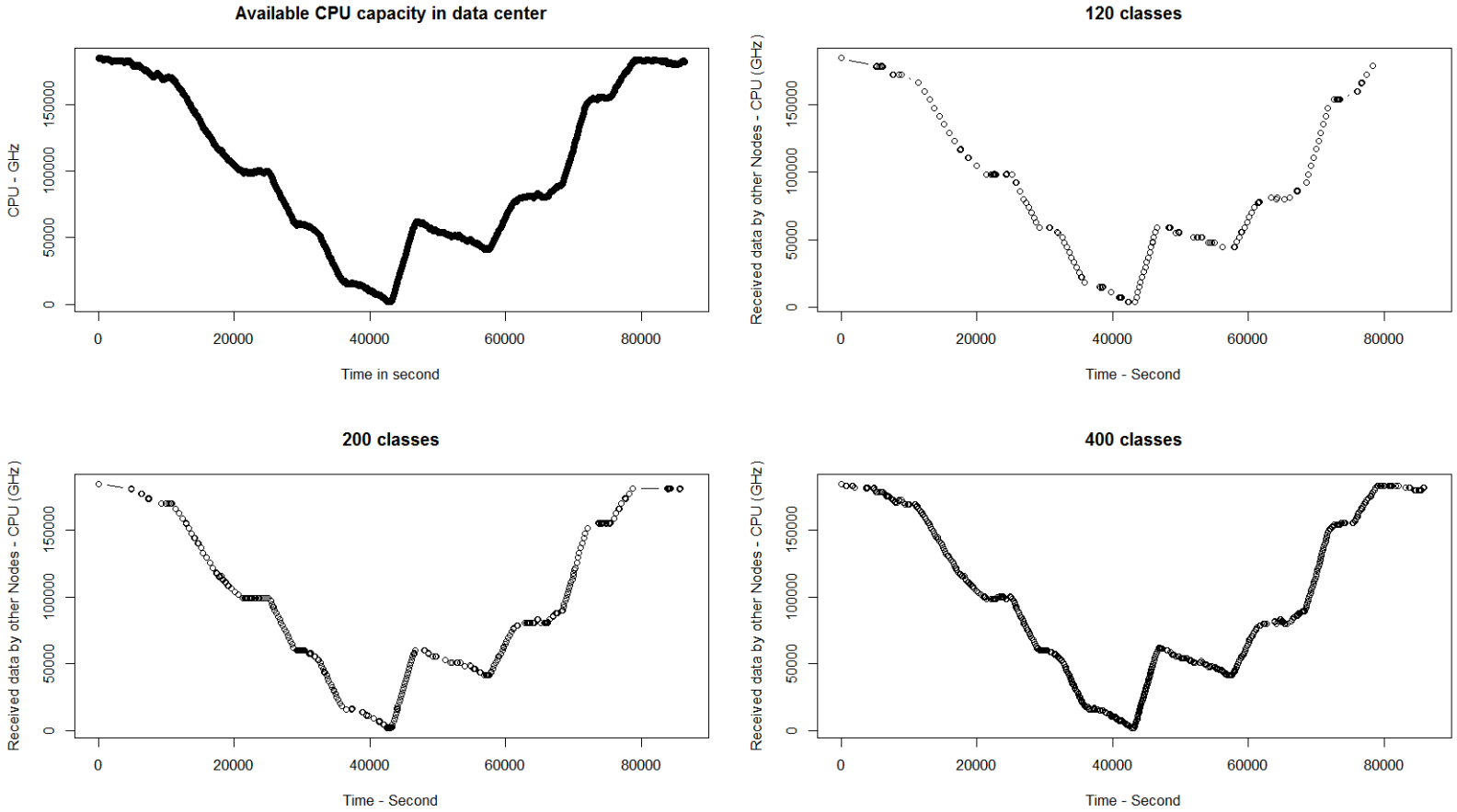

Figure 5.6: Equal-sized class-based update policy. How a cloud management system views cloud resources with different number of classes (N_{eq}) values 120, 200, and 400. The updates points are plotted a circles in each graph.

5.3.2 Exponential-sized classes

When a growth factor "f" is greater than 1 (see the equation 2.4 on page 32), we will have an exponential-sized class-based update policy. In this update policy, the base class size finds out by a base factor "β" and the classes size increases geometrically by a factor "f". In an exponential-sized class-based update policy, when a growth factor "f" decreases and/or a base factor "β" decreases, the total capacity "C" divides in to more classes, and as a result more updates expected due to the simulated data center's resource changes. In order to find an appropriate base factor "β" and growth factor "f" value, the exponential-sized class-based update policy with different "β" value and "f" values was applied to the data center model which was described in section 4.1. Figure 5.7 shows how these two factors effect on the number of updates due to the data center's CPU capacity changes.

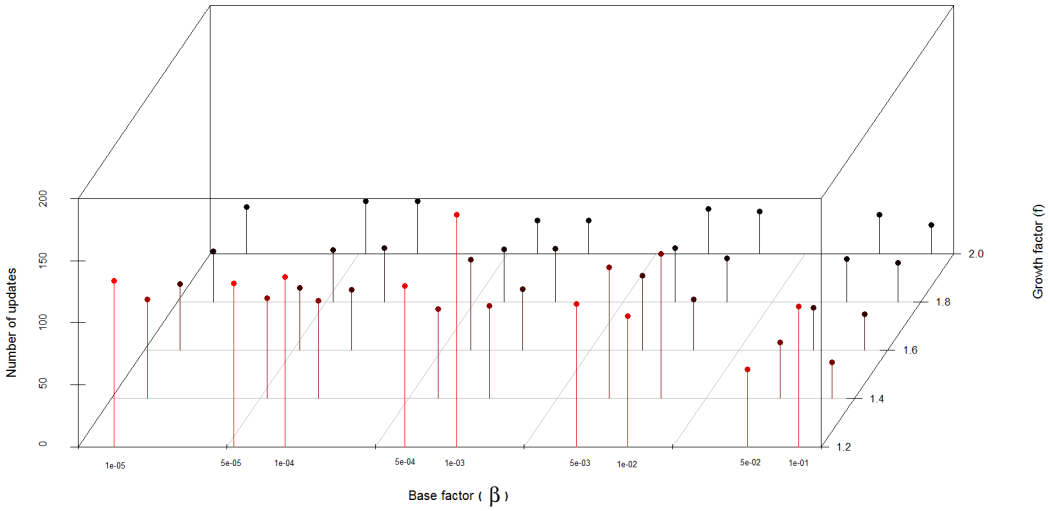

Figure 5.7: Exponential-sized class-based update policy. Number of updates per different values for growth factor "f" and base factor β. The "X" (base factor) axis is in logarithm scale.

As discussed earlier in this section, fewer updates are expected when a growth factor "f" increases. The experiment illustrates abnormal behavior for some growth factor values. Figure 5.8 shows the number of updates for base factor values 0.000001, 0.00001, 0.01, and 0.1 with different growth factor values. In some points, which are pointed in red, more updates observed when the growth

factor was increased.

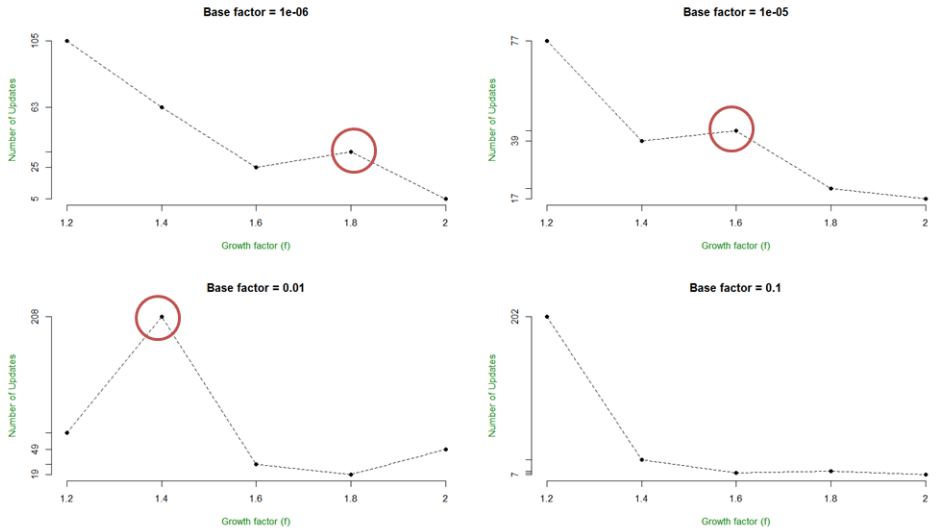

Figure 5.8: Exponential-sized class-based update policy. Number of updates for base factor values 0.000001, 0.00001, 0.01, and 0.1 with different growth factor values.

As discussed earlier, in a class-based update policy a new update triggers when the available capacity passes a class boundary, As a result, it can generate lots of unnecessary updates when the available resources fluctuates around a class boundary. Figure 5.9 visualizes this problem for an exponential-sized class-based update policy where $f = 1.4$ and $\beta = 0.01$. The periods that this update policy generates unnecessary updates are pointed in red. This explanation illustrates the reason of unexpected behavior in Figure 5.8. The hysteresis method can be used to to prevent excessive unnecessary updates. In this method, a new update triggers when not only the available capacity passes the class boundary but also the magnitude of the changes are significant (e.g., the available capacity passes a middle of adjacent classes) [78].

The number of updates is not a sufficient factor to make a decision on a growth factor and a base factor values. It is also important to consider how the receiver node(s) (e.g., CMS) in an OSPF domain views the data center's resources. The analysis results enable us to see how the update policy reacts to changes in the data center's resources and how well updates can describe the data center's resources. Figure 5.10 and Figure 5.11 show how CMS views a data center's available CPU capacity with different values for base factor and growth factor. The complete set

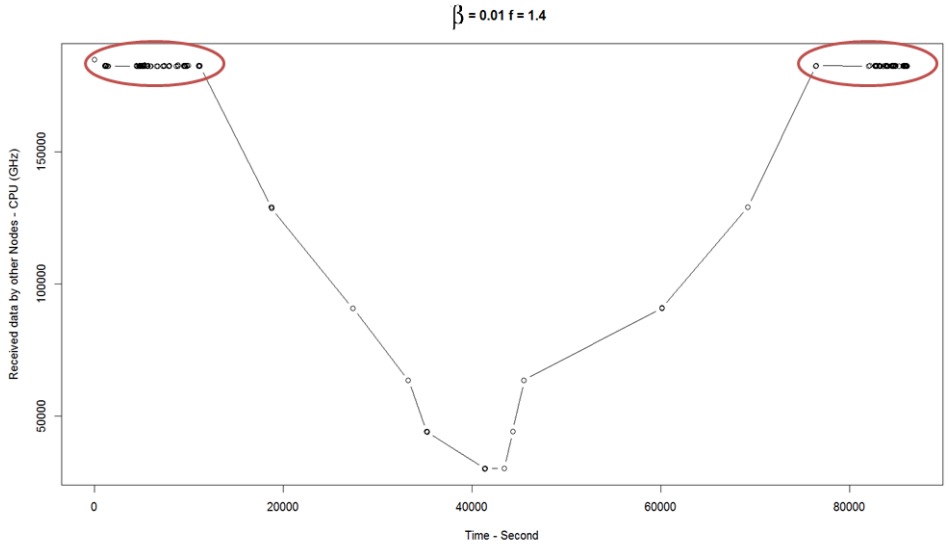

Figure 5.9: Exponential-sized class-based update policy. How the update policy send updates due changes in CPU value when base factor "β" = 0.01 and growth factor "f" = 1.4. The periods pointed in red show the duration that this policy generates unnecessary updates.

of result are given in Appendix F. These Figures enable us to see the inaccuracy of the resource management system's view over the state of the cloud resources and/or unnecessary updates in some period.

In an exponential-sized class-based update policy updates are sent more frequently when there are few available resources. This phenomenon occurs due to smaller class size in situations that we have fewer amounts of available resources. By this means this policy can provide more accurate information when we have limited amount of resources in a data center. This approach is beneficial for sending a data center resource - because the CMS must be more careful in managing available resources when they are limited.

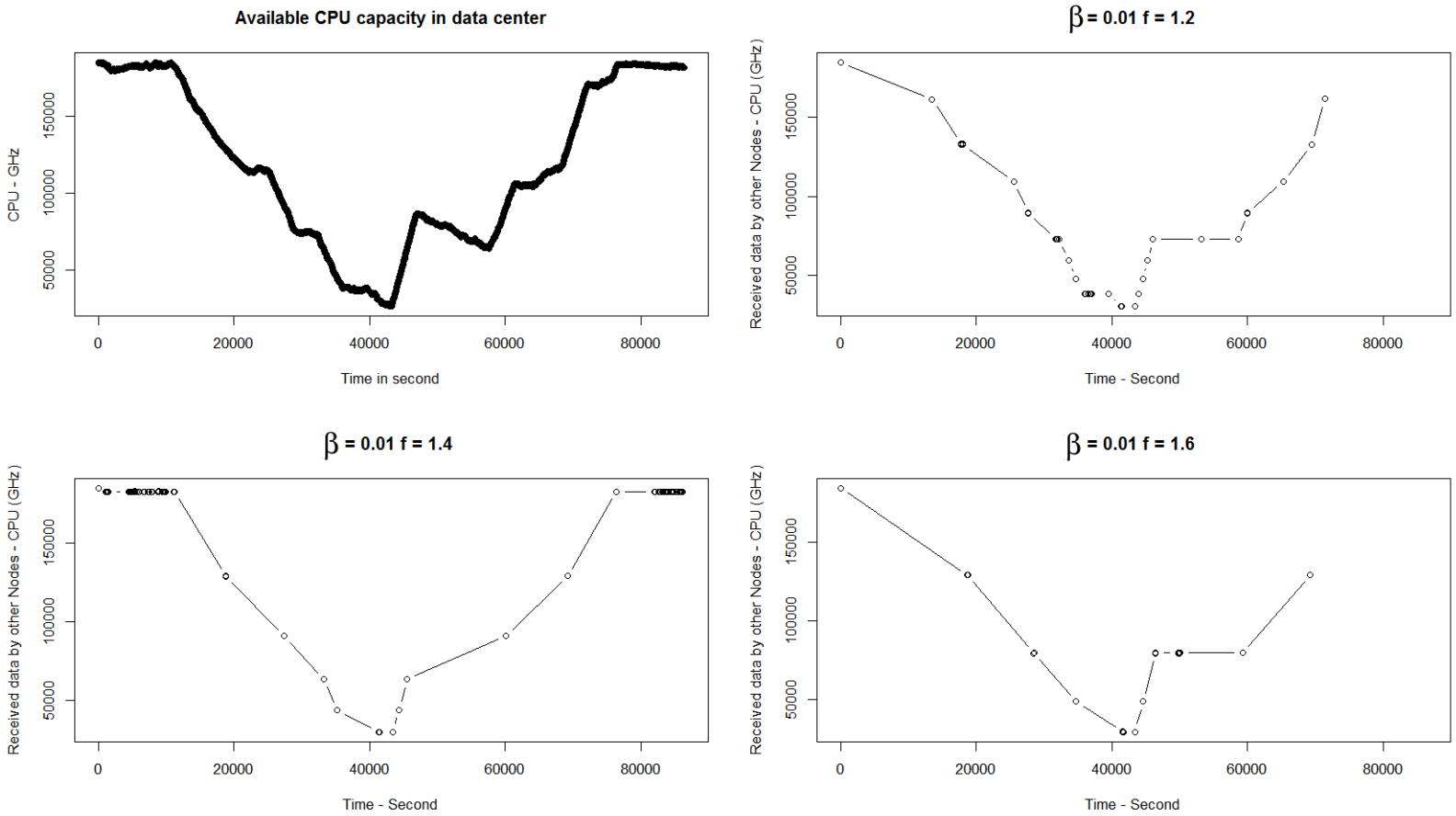

Figure 5.10: Exponential-sized class based update policy. How a cloud management system views data center's CPU resources with fix base factor ($\beta = 0.001$) but different growth factor "f" values, the updates points are plotted a circles in each graph.

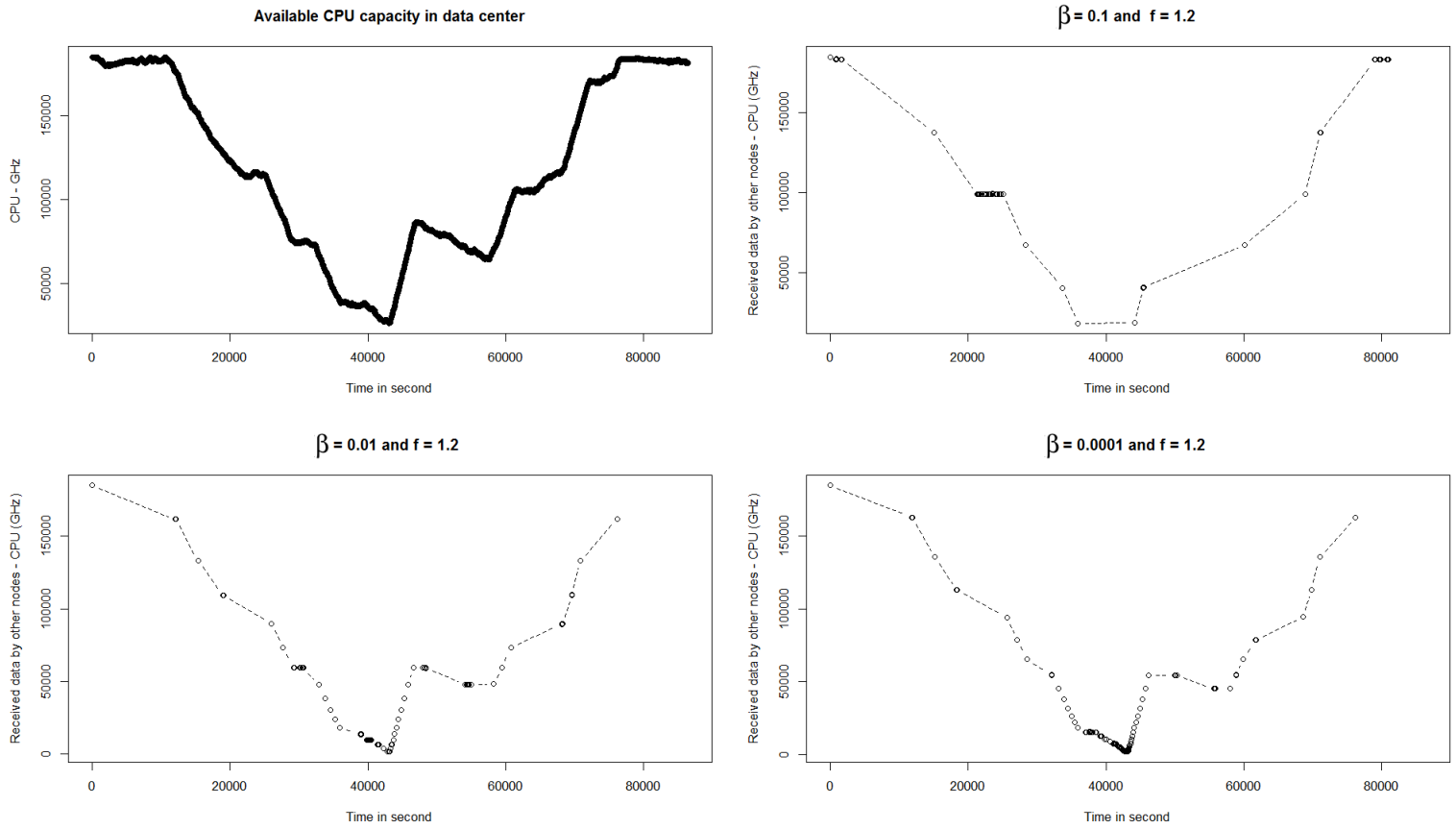

Figure 5.11: Exponential-sized class based update policy. How a cloud management system views data center's CPU resources with a fix growth factor ($f = 1.2$) but different base factor "β" values, the updates points are plotted a circles in each graph.

5.4 Threshold-based update policy

The threshold-based update policy triggers an update when the change in a resource's availability exceeds a certain threshold value. In this policy the changes in a resources can be an absolute changes or a relative changes.

5.4.1 *Absolute* threshold-based update policy

The threshold-based with absolute change update policy triggers an update when the absolute change in a resource's availability exceeds a certain threshold value as described by equation 2.2 on page 32. In order to find an appropriate threshold value, an absolute threshold-based policy with different threshold values was applied to the data center model which was described in a section 4.1. Figure 5.12 shows how the threshold value effects on the number of updates due to the data canter's CPU capacity changes.

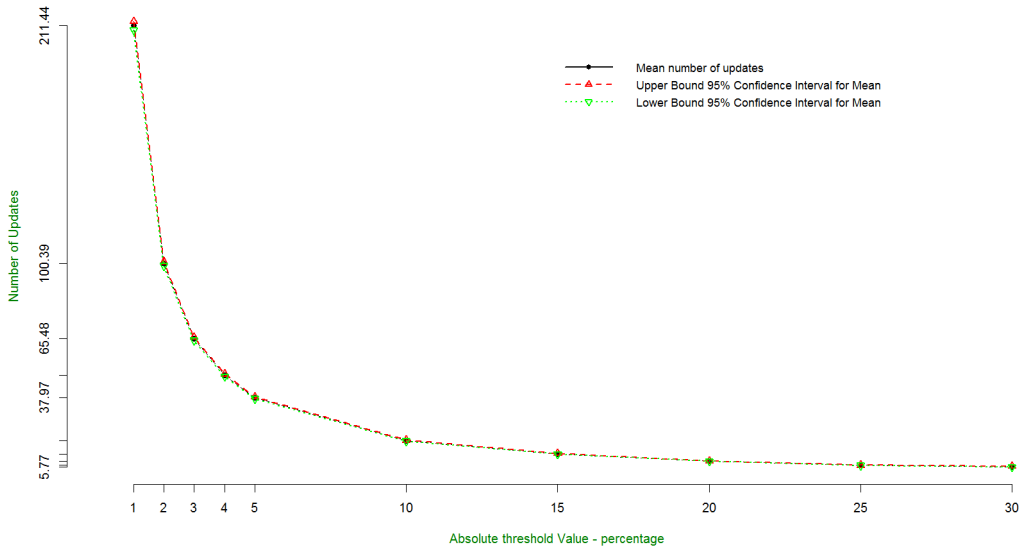

Figure 5.12: *Absolute* threshold-based update policy. Number of updates due to CPU changes per different threshold values. The "X" axis are in logarithm scale.

This experiment clearly illustrates the hypothesis that the higher threshold value results in sending fewer updates than dose a smaller value. While reducing the number of Cloud LSA updates will reduce the network load, there is a tradeoff between the update frequency and accuracy of cloud resource management system information about cloud resources.

The number of updates is not a sufficient factor to determine a threshold value. It is also important to consider how the receiver node(s) (e.g., CMS) in an OSPF domain views the data center's resources. The analysis results enable us to see how the update policy reacts to changes in the data center's resources and how well updates can describe the data center's resources. The result for threshold values 2%, 3%, and 4% are shown in Figure 5.13. The complete set of results are given in Appendix G. These Figures enable us to see the inaccuracy of the resource management system's view over the state of the cloud resources in some period of the time. The experiments show this policy behaves a lot like an equal-sized class-based update policy.

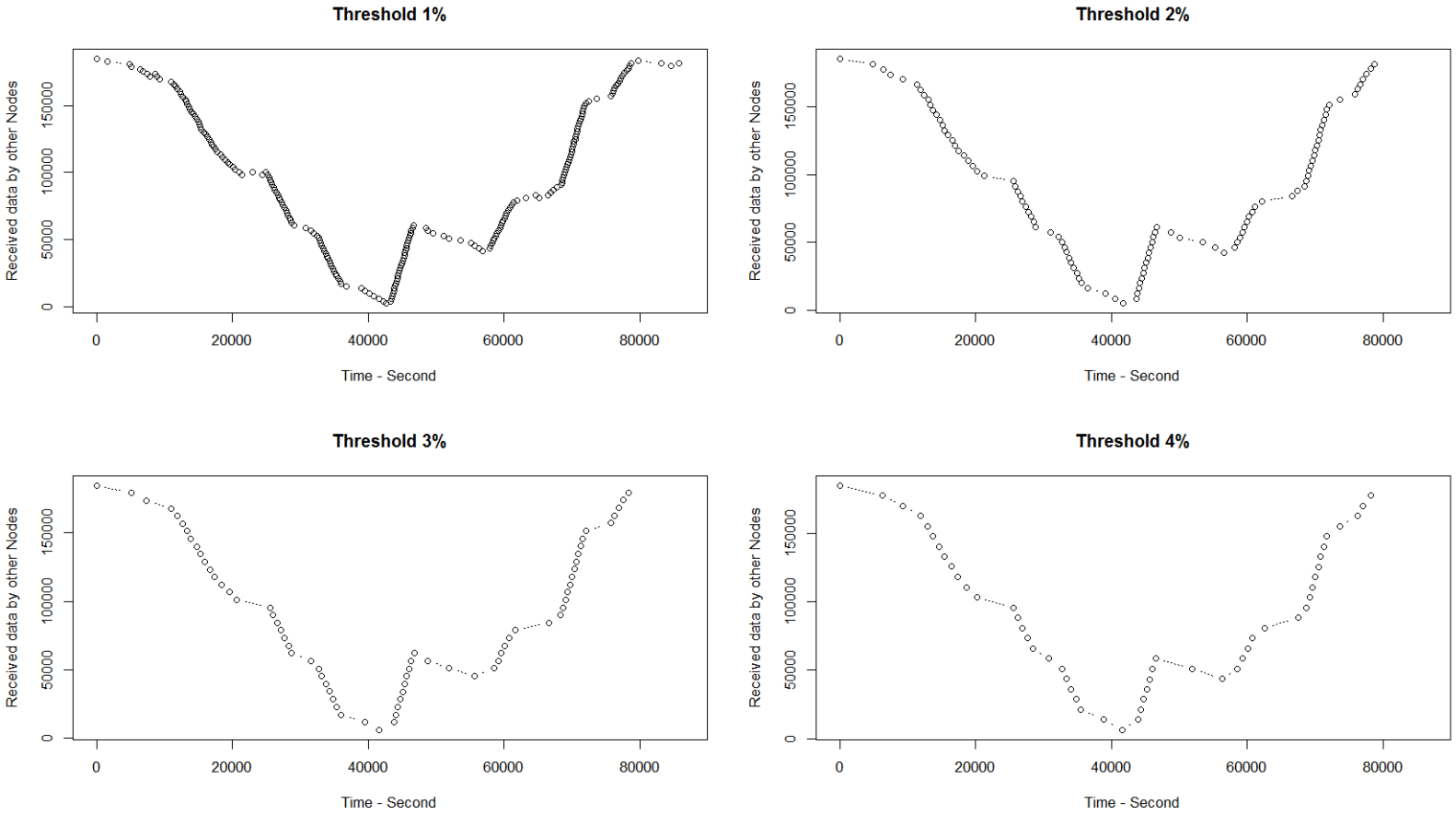

Figure 5.13: *Absolute* threshold-based update policy. How a cloud management system views cloud resources with different threshold values 2, 3 and 4 %, the updates points are plotted a circles in each graph.

5.4.2 *Relative* threshold-based update policy

The threshold-based with *relative* change update policy triggers an update when the relative change in a resource's availability exceeds a certain threshold value based on equation 2.3 on page 32. In order to find an appropriate threshold value, a relative threshold-based update policy with different threshold values was applied to the data center model which was described in section 4.1. The threshold was set to different values between 5% and 100% in increments 5%. Figure 5.14 shows how the threshold value effects on the number of updates based on the data center's CPU capacity.

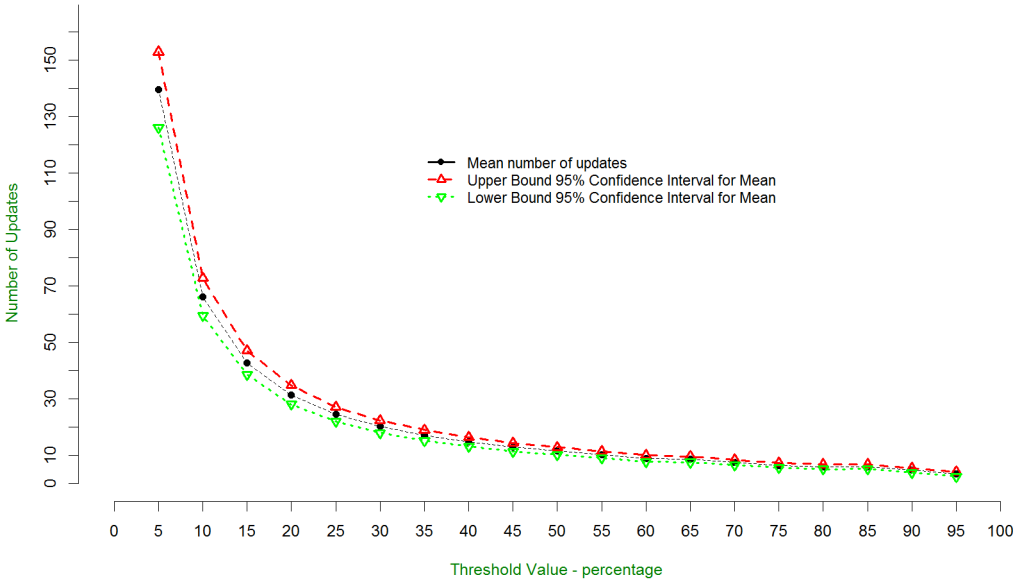

Figure 5.14: *Relative threshold-based* update policy. Number of updates per different threshold values based on changes of a data center's CPU capacity.

This experiment clearly illustrates the hypothesis that the higher threshold value results in sending fewer updates than dose a smaller value. While reducing the number of Cloud LSA updates will reduce the network load, there is a tradeoff between the update frequency and accuracy of cloud resource management system information about cloud resources.

The number of updates is not a sufficient factor to determine a threshold value. It is also important to consider how the receiver node(s) (e.g., CMS) in an OSPF domain will views the data center's resources. The analysis results enable us to see how the update policy reacts to changes in the data center's resources and how well updates can describe the data center's resources. The result for threshold value of 5%, 20%, and 50% are shown in Figure 5.15. The complete set of results are given in Appendix H. These Figures enable us to see the inaccuracy of the resource management system's view over the state of the cloud resources in some period of the time.

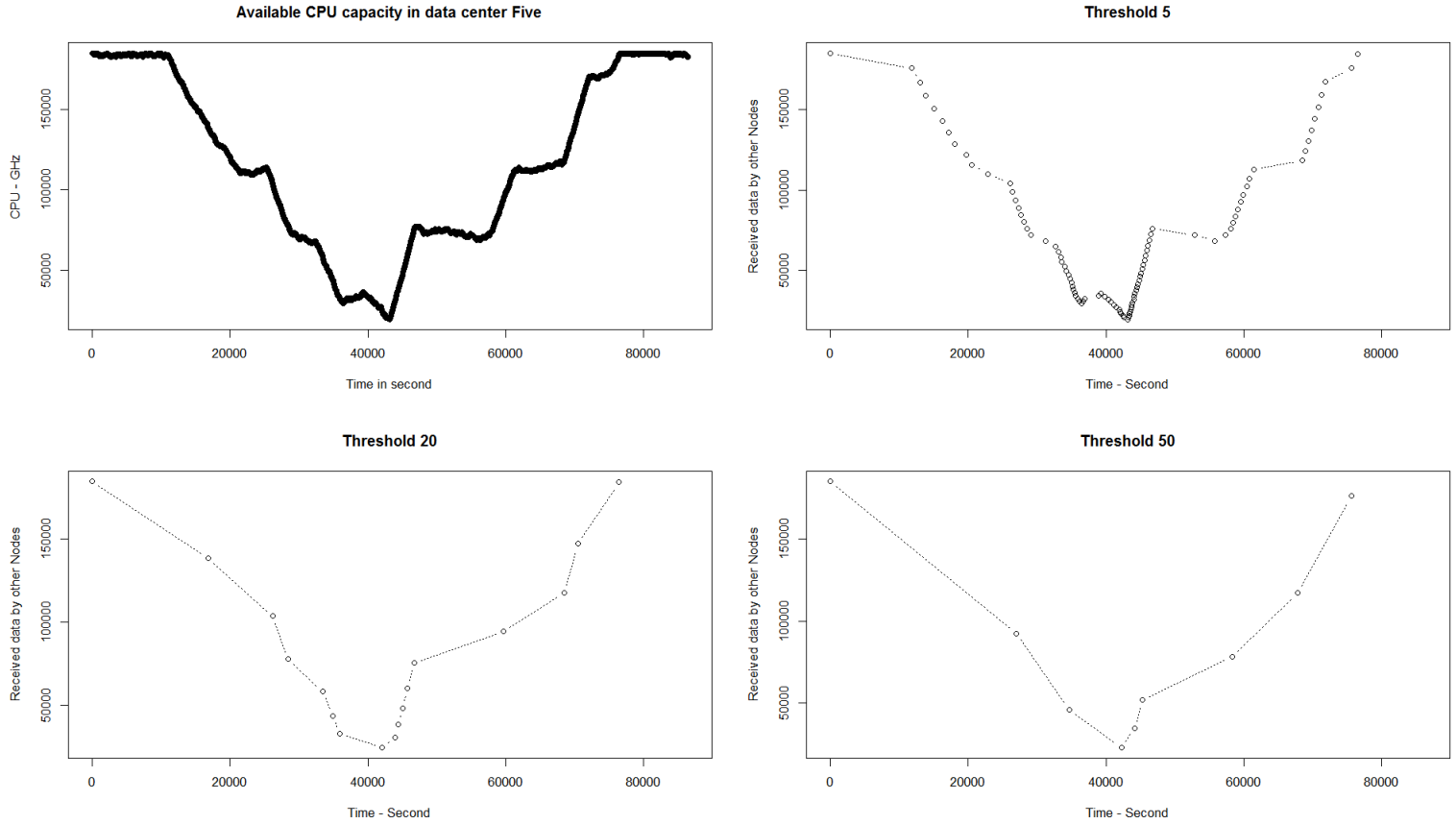

Figure 5.15: *Relative* threshold-based update policy. How a cloud management system views cloud resources with different threshold values 5, 20 and 50 percent, the updates points are plotted a circles in each graph.

5.5 Summary

A *periodic update policy* periodically triggers a new LSA update without considering the extent of available resources. As a result, a significant change can be ignored and unnecessary LSA updates can occur often. On the other hand, the *immediate update policy* generates a LSA update after every time there is a change in resources. This policy maximizes the accuracy of resource management, but dramatically increases the network load. As the type and rate of resource request changes during the day, the available cloud resources fluctuate. In conclusion, periodic update and immediate update policies do not allow update policy to adapt to the availability of the data center's resources.

A *threshold-based policy* regarding what is a relevant change and a class-based policy with exponential-sized class update try to avoid unnecessary updates by sending an update only when a *significant* change in resources occurs. Also, for both these update policies, updates are sent more frequently when there are few available resources - hence the CMS must be more careful in managing them. Such characteristics make either of them a good candidate for this project.

The studies show that determining the appropriate parameters for the policy is more influential on the amount of traffic and accuracy than the choice of the policy itself [73]. For the purposes of this thesis project we will use a threshold-based policy concerning relevant change and we try to determine the correct threshold values for each type of resource.

Chapter 6

Analysis

As discussed earlier in a section 3.3, the Cloud LSAs distribute the cloud resource information, thus providing the data required by the CMS to allocate resource in the embedded data centers. It is clear that adding new Opaque LSA into OSPF-TE results in an increase in the amount of OSPF protocol traffic. The Cloud LSA can be described by a number of data center's related parameters (e.g., the available CPU, RAM, storage, etc.). It is worth emphasizing that the amount of opaque data carried in Cloud LSAs directly increases the OSPF protocol traffic overhead in network. However, more data carried in Cloud LSAs provides more information for the CMS which can result in a more accurate decision, but it also results in a more overhead due to the Cloud LSAs passing over the network.

The proposed Cloud LSA in this thesis project carries four parameters: available CPU, RAM, storage, and data center's location. As shown in the Figure 3.6 on page 49, the proposed Cloud LSA has 44 bytes of data. In order to send an OSPF LSA into the OSPF network as a link-state update (LSU), the LSU header (28 bytes), IP header (20 bytes), and Ethernet header and trailer (18 bytes) [if applicable] should be added to Cloud LSA. As a result, sending *one* Cloud LSA link-state update loads the network with an additional 110 bytes, see Table 6.1.

Table 6.1: Cloud LSU parts and sizes.

LSU parameters	**Size*(Byte)***
Cloud LSA	44
LSU Header	28
IP header	20
Ethernet header and trailer	18
Total size	**110**

Additionally, each LSU has an acknowledgment response. The link-state acknowledgment packet consists of the OSPF header (20 bytes) plus the link-state advertisement header (24 byte). In order to send an acknowledgment packet into the OSPF network, IP header (20 bytes), and Ethernet header and trailer (18 bytes) [if applicable] should be added to this packet. As a result, sending *one* link-state acknowledgment loads the network with an additional 82 bytes, see Table 6.2.

Table 6.2: Link-state acknowledgment parts and sizes.

Ack parameters	**Size*(Byte)***
LSA Header	24
OSPF Header	20
IP header	20
Ethernet header	18
Total size	**82**

As a conclusion, when an OSPF router originates *one* LSU containing the Cloud LSA the network loads with an additional 192 bytes (in a sender point of view) as shown in Table 6.3.

Table 6.3: Extra Load by sending a Cloud LSA.

parameters	**Size*(Byte)***
Cloud LSA	110
Acknowledgment	82
Total size	**192**

6.1 Performance of solution

As described in section 4.2, the Cloud-OSPF extension to OSPF-TE invokes the OSPF router to send an LSU if a significant change occurs in the data center's resources. The link-state update policy is responsible for deciding which changes are considered significant, see section 2.5. To evaluation the performance of the proposed solution several test scenarios were utilized. The test scenarios are described in next section. It is clear that the test results will be biased based on data center model (presented in section 4.1), the update policy algorithm (presented in section 5), and the test bed topology (given in Figure 6.2). In this part of the evaluation, the worst case protocol overhead is calculated without considering the data center model and update policy algorithm in a fully meshed connected network topology with "N" number of OSPF nodes.

However, to analyze the Cloud LSA protocol traffic overhead it is not sufficient to consider the amount of traffic that proposed Cloud-OSPF extension sends into an OSPF network. As discussed in section 2.4, based on LSA N-squared problem, in an OSPF network with N OSPF routers, flooding each link-state update produces at maximum $O(N^2)$* redundant LSAs. As a result, to analyze the proposed solution protocol overhead the traffic loaded in OSPF network by flooding Cloud LSAs will be considered (and *not* a traffic that a Cloud-OSPF injects into the OSPF network). The loads due to Cloud LSAs in the OSPF network (in receivers point of view) can be calculated using equation 6.1.

$$Extra\ loads = D \times R_{Netwok} \times N_{CloudLSA} \times Size_{CloudLSA} \tag{6.1}$$

Where "D" denotes the number of embedded data centers, "R_{Netwok}" denotes number of redundant LSAs that OSPF network generates for flooding a LSA, "$N_{CloudLSA}$" denotes number of Cloud LSAs that Cloud-OSPF injects into the OSPF network, and "$Size_{CloudLSA}$" denotes a Cloud LSA size.

Consider a fully meshed connected network with "N" OSPF router and *one* embedded data center. The Cloud-OSPF extension sends a Cloud LSA update when a data center's resource state changes. The update policy module of Cloud-OSPF extension is responsible to take decisions when the CMS need to be updated with the latest data center's resource information. As described in section 2.4, the OSPF router can produce a maximum of one LSA in each MinLSAInterval period of 5 seconds. As a result, using any type of update policies, the maximum rate for sending updates by OSPF router, is limited to *one* LSU per 5 seconds.

*Order of N^2

Figure 6.1 shows the relation between the rate of sending Cloud LSAs and loads that *sender* injects in to the network embedded cloud due to Cloud LSAs. In this graph, we add zero line as a base (i.e., red color). Additionally, this graph shows by increasing the interval period the Cloud LSAs loads rapidly decreases and inclines to zero.

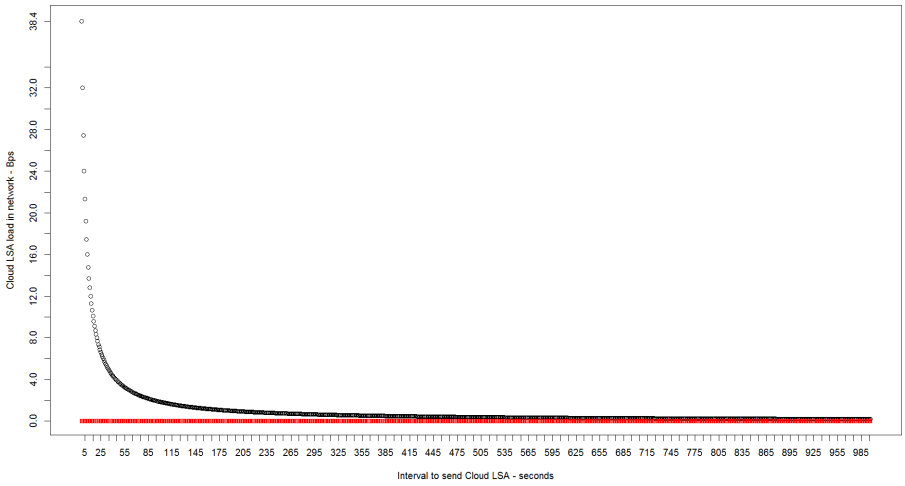

Figure 6.1: The Cloud LSA traffic that a Cloud-OSPF sender sends in a network embedded cloud for different interval values. The rates also limited by MinLSInterval (i.e., 5 seconds).

Thus, the proposed solution in this thesis work can handle cases that a data center's resource changes in such a way that a CMS need to be updated at a minimum rate of 5 seconds interval. Hence, the Cloud-OSPF sender sends at most 192 *bytes per 5 seconds* or 38.4 *bytes per second* (Bps). Considering LSA N-squared problem, in this case the network loads by at maximum $O(N^2) \times 192$ *bytes per 5 seconds* or $O(N^2) \times 38.4$ *bytes per second* (Bps).

This project concerns a network embedded cloud, which consists of a number of distributed data centers. To provide the latest information to the CMS an extended OSPF routers at each data center needs to send Cloud LSAs. As a result, the number of LSUs which send in a network is a direct function of the number of data centers in the OSPF network. In a fully meshed connected network with "D" embedded data centers and "N" OSPF routers the average traffic due to Cloud LSAs in receiver point of view will be $(D \times O(N^2) \times 38.4)$ Bps. If each "D" data centers send *one* LSU at the same time, then the peak overhead in traffic caused

by this protocol will be $(D \times O(N^2) \times 192)$ bytes every 5 seconds.

However, OSPF and its flooding protocol are applicable for small and medium sized network because of the issue of the LSA N-squared problem. Studies show the maximum number of OSPF routers per AS is limited to around 50 [9, 79, 80]. As a consequence, if the network is fully-meshed and the number of OSPF routers limits to 50, and if beside each OSPF routers there is an embedded data center, the maximum average of additional protocol overhead can be calculated as around 4.610* Megabytes per second (MBps) with a peak of around 23.050† Megabytes (MB) per 5 seconds (see equation 6.1, equation 2.1 on page 31, and Table 6.3).

As the network embedded cloud is not fully deployed, there is no clear assumption about the number of embedded data centers in this cloud model. However, the telecommunication providers, to enable cloud functionality, can install compute resources (i.e., racks with some servers) beside each of their network nodes. As a result, the number of embedded data center in a network embedded cloud can be much more than 50. Summary of this theoretical analysis is shown in Table 6.4.

Table 6.4: Cloud LSA worst average overhead and worst peak overhead analysis. The "N" represents a number of OSPF routers and the "D" represents a number of embedded data centers.

Number of OSPF nodes	**Number of embedded data centers**	**Peak Overhead in bytes**	**Avg Overhead Bps**
N	1	$O(N^2) \times 192$	$O(N^2) \times 38.4$
50	50	23,049,600	4,609,920
N	D	$D \times O(N^2) \times 192$	$D \times O(N^2) \times 38.4$

6.2 Evaluation based on test scenarios

In this section the description related to set up a test environment is given. Each test scenarios were run 20 time to have results with a confidence interval (CI) of 95%. Thereafter, the results from one particular test round is given to have a better

*The exact value is 4609920 Bps.
†The exact value is 23049600 Bytes per 5 seconds.

understand the behavior of proposed solution based on selected update policy (i.e., relative threshold update policy) and proposed data center model.

6.2.1 Set up test environment

The test machine is a HP EliteBoook 8560p with an *"Intel® Core™i7-2620M (2.70 GHz, 4 MB L3 cache)"* processor. It is a dual-core processor with hyper-threading [81], which provides 4 effective CPUs. Also, the system has 8 GB of DDR3 (1333 MHz) SDRAM.

The test machine's operating system (OS) is Ubuntu Linux*. It uses the Oracle VM VirtualBox program† [82] to create a network topology for testing. VirtualBox hosts six virtual machines (VMs) as OSPF routers. Each VM's specifications are shown in Table 6.5.

Table 6.5: VMs specification on test bed.

Base memory	1024 MB
Chip-set	PIX3
Number of CPU	1
Network Interface	Intel PRO/1000 T server

Each VM is equipped with an instance of the Quagga software in order to utilize an OSPF router‡. The Quagga routers are interconnecting using VirtualBox's internal networking component. The Quagga routers are interconnected by a simulated link and configured in area 0 (as shown in Figure 6.2).

Three different scenarios were used for testing. These scenarios all have the same topology, but different numbers of embedded data centers were used - as shown in Figure 6.3, Figure 6.4, and Figure 6.5. The OSPF protocol overhead is measured in order to determine the system's performance. The traffic on each links was captured by *Wireshark*§ application [83]. In order to analyze the collected data, the *"R"* statistics application [84], *Microsoft Excel*, and a shell script were used.

*Ubuntu release 12.04 (precise) 64bit, kernel Linux 3.2.0-33-generic, and GNOME 3.4.2.
†Oracle VM VirtualBox Version 4.1.12_ubunto
‡The Quagga version is 0.99.21
§Version 1.8.3 (SVN Rev 45256 from /trunk-1.8)

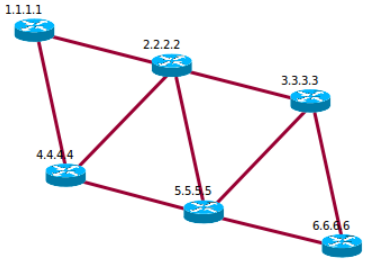

Figure 6.2: The test bed topology with the router ID of each of the Quagga routers.

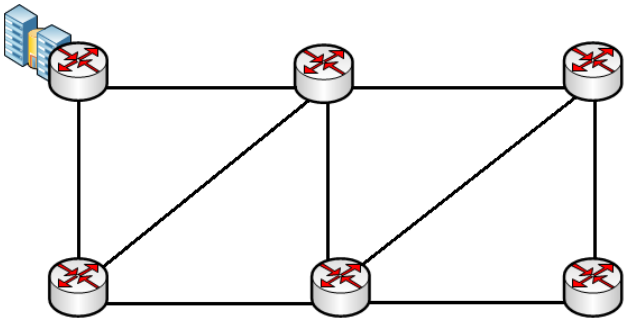

Figure 6.3: First test scenario - One embedded data centers in the network.

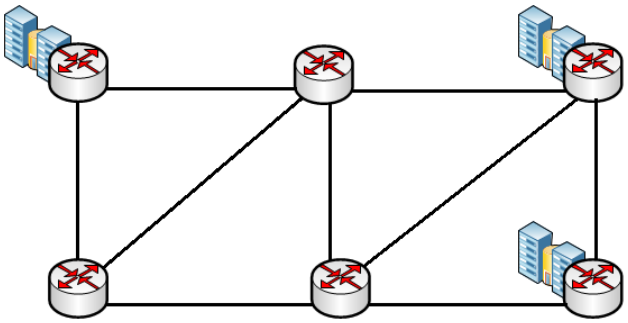

Figure 6.4: Second test scenario: Three embedded data centers in the network.

As described earlier the two key factors, which can effect on the evaluation, are the update policy and resource modeling of each data center. Based on the analysis in Chapter 5, and the complete set of results in Appendix D, Appendix E, Appendix F, Appendix G, and Appendix H, the relative threshold based update policy with threshold value of 20% was selected for this project. MinLSInterval was left at its default value (i.e., 5 seconds). The interval value for checking the data center's resources is set to same value as MinLSInterval (i.e., 5 seconds). In

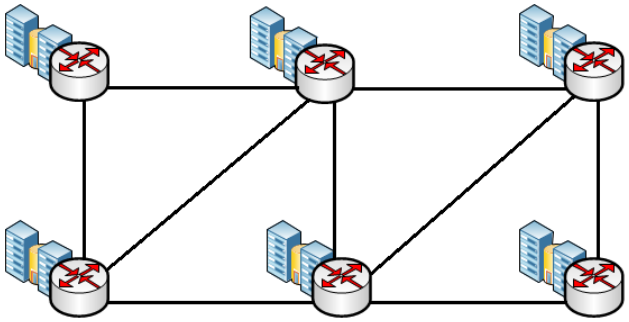

Figure 6.5: Third test scenario: Six embedded data centers in the network.

this case the Cloud-OSPF module will check the data center's resources every 5 seconds and then it will decide if it should send out a LSU. Therefore, unnecessary changes in the data center's resource are ignored. Additionally, the data center model described in the section 4.1 is used in the three test scenarios. To utilize a connection between Cloud-OSPF and Quagga's OSPF API the port number 8898 * is used.

It is worth mentioning that the OSPF-Sender module in test scenarios sends a new Cloud LSA when the relative changes in available CPU capacity or RAM capacity or storage capacity exceeds the threshold value, see Algorithm 1.

Algorithm 1 Logic which is used in a Cloud-OSPF module to send a new Cloud LSA.

```
if (CPU relative change ≥ Threshold Value, OR
RAM relative change ≥ Threshold Value, OR
CPU relative change ≥ Threshold Value) then
   Send a new Cloud LSA
end if
```

6.2.2 Expected result

As discussed in section 2.4, LSA flooding in an OSPF network has N-squared problem when a OSPF network is fully meshed. The test bed network in this work is not a full mesh network (see Figure 6.2), so we estimates how many redundant messages required to flood a Cloud LSA. The test bed topology contains 6 OSPF

*This port number is not assigned by IANA [85].

routers that provide entirely 18 network interfaces. Suppose one of the OSPF routers injects a Cloud LSA into the network. Consider to the OSPF reliable flooding (see section 2.4), other OSPF routers in the network (i.e., 5 routers) receive that LSA and then they forward it to all of their interfaces except the one where that LSA was received from it. As a result, in the test bed network, sending one Cloud LSA produces 13* instances of LSAs.

As discussed above, the Cloud-OSPF module sends a new cloud LSA using a logic discussed in an Algorithm 1. Additionally, Chapter 5 described how the number of updates can be estimated for the proposed data center model. The expected number of updates based on Algorithm 1 for relative threshold update policy with threshold value 20% is shown in Table 6.6. A complete set of results are given in Appendix I.

Table 6.6: Relative threshold-based update policy. Expected number of updates for a relative threshold value 20% due to changes in a simulated data center's resources based on a logic described in an Algorithm 1.

Mean		39
95% CI for Mean	Lower Bound	34
	Upper Bound	43
Median		29
Variance		574
Std. Deviation		23.97
Minimum		16
Maximum		145

Finally, using equation 6.1 and the information in Table 6.3, Table 6.6, and a expected number of redundant LSAs in a test bed network (i.e., 13 LSAs), the expected Cloud LSAs traffic for test scenarios with 6 OSPF router can be calculated as shown in Table 6.7.

*Number of redundant messages = All network interfaces in a network - (Number of routers - 1)

Table 6.7: Expected Cloud LSAs traffic for three scenarios.

Scenario	number of data center	Expected traffic - Bps				
		Mean	Lower Bound	Upper Bound	Min	Max
1	1	1.127	0.982	1.242	0.462	4.189
2	3	3.380	2.947	3.727	1.387	12.567
3	6	6.760	5.893	7.453	2.773	25.133

6.2.3 Measured results

Each test scenario was executed 40 times to have results with a confidence interval of 95%. Each round of testing simulated 24 hours. In each of these scenarios the testing starts at midnight, i.e., 00:00.

In test scenarios, the network state was remained stable, which means *no* link(s) or node(s) failure occurred in test periods. In this case, the pure OSPF protocol overhead was observed as 205.453 ± 0.904 Bps (see Table 6.8).

Table 6.8: Summery of captured *pure* OSPF traffic in Bps unit.

Mean		205.453
95% CI for Mean	Lower Bound	203.207
	Upper Bound	207.699
Median		205.626
Variance		0.817
Std. Deviation		0.904
Minimum		204.475
Maximum		206.258

In a first test scenario with *one* embedded data center, the statistical analysis on captured OSPF traffic shows the proposed solution produces 1.214 ± 0.885 Bps protocol overhead (see Table 6.9). In this case the proposed solution adds at *mean* 0.590 ± 0.430% extra load to pure OSPF protocols.

Table 6.9: First test scenario. Summery of captured *Cloud LSAs* traffic in Bps unit.

Mean		1.214
	Lower Bound	0.800
95% CI for Mean		
	Upper Bound	1.629
Median		0.866
Variance		0.784
Std. Deviation		0.885
Minimum		0.554
Maximum		3.537

In a second test scenarios with *three* embedded data centers, the statistical analysis on captured OSPF traffic shows the proposed solution produces 3.658±1.095 Bps protocol overhead (see Table 6.10). In this case the proposed solution adds at *mean* 1.780±0.533 % extra traffic to pure OSPF protocols.

Table 6.10: Second test scenario. Summery of captured *Cloud LSAs* traffic in Bps unit.

Mean		3.658
	Lower Bound	2.816
95% CI for Mean		
	Upper Bound	4.500
Median		3.505
Variance		1.200
Std. Deviation		1.095
Minimum		2.269
Maximum		5.299

In a third test scenarios with six embedded data centers, the statistical analysis on captured OSPF traffic shows the proposed solution produces 7.024±1.325 Bps protocol overhead (see Table 6.11). In this case the solution adds at *mean* 3.419 ± 0.565 % extra traffic to pure OSPF protocols.

Table 6.11: Third test scenario. Summery of captured *Cloud LSAs* traffic in Bps unit.

Mean		7.024
95% CI for Mean	Lower Bound	6.725
	Upper Bound	7.323
Median		7.567
Variance		1.755
Std. Deviation		1.325
Minimum		4.333
Maximum		8.839

Using equation 6.1, equation 2.1 on page 31, and the analysis on Table 6.3, the maximum expected additional load due to Cloud LSAs in a fully-meshed connected network topology with 6 OSPF router can be calculated as: 960 Bps for a case that we have *one* embedded data center, 2880 Bps for a case that we have *three* embedded data centers, and 5760 Bps for a case that we have *six* embedded data centers.

In a worst expected load calculation, the network topology considered as a fully-meshed connected while the test bed topology was not fully meshed connected. As a result we experienced less number of redundant updates (13 redundant LSAs in a network due to inject one LSA) than in full meshed topology. Additionally, we did not consider the update policy and data center model in worst expected load calculation, and we assumed the Cloud LSAs floods every MinLSInterval. These factors can reduce the number of LSAs in the network and leads to the biased results based on network topology, data center model, and update policy.

Figure 6.6 shows the results from test scenarios has a correlation with expected traffic. It is worth mentioning that based on OSPF protocol LSRefreshTime timer (see Table 2.6 on page 30) if the extended OSPF router dose not generate the new Cloud LSA within this interval the OSPF router will re-originate the Cloud LSA. That is why the measured Cloud LSAs traffic are more than mean expected Cloud LSAs traffic. To have more reliable data with lower variance value we need to run the test scenarios more than 40 times, which lefts as a future work.

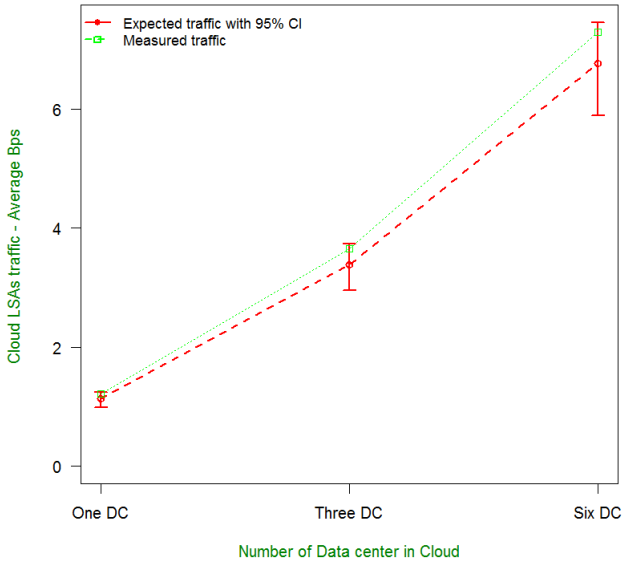

Figure 6.6: Comparison between expected and captured Cloud LSAs traffic.

6.2.4 More observations in testing

In this section we look in to the test results for three scenarios for *one* round. This section will enable us to analyze the behavior of the proposed solution. Figure 6.7 shows the OSPF protocol traffic as measured by Wireshark application for a second test scenario. The graphs set to shows the OSPF traffic in **bytes per 10 minute interval**. We shows our results in the bytes per 10 minute interval because each scenario contained the network traffic for 24 hours (i.e., 86400 seconds). As a result the graphs with the unit of Bps was not clear and the viewer was not able to distinguish between different traffic patterns. The complete set of results for other test scenarios, and also the results with a scale of **bytes per minute** are given in Appendix J.

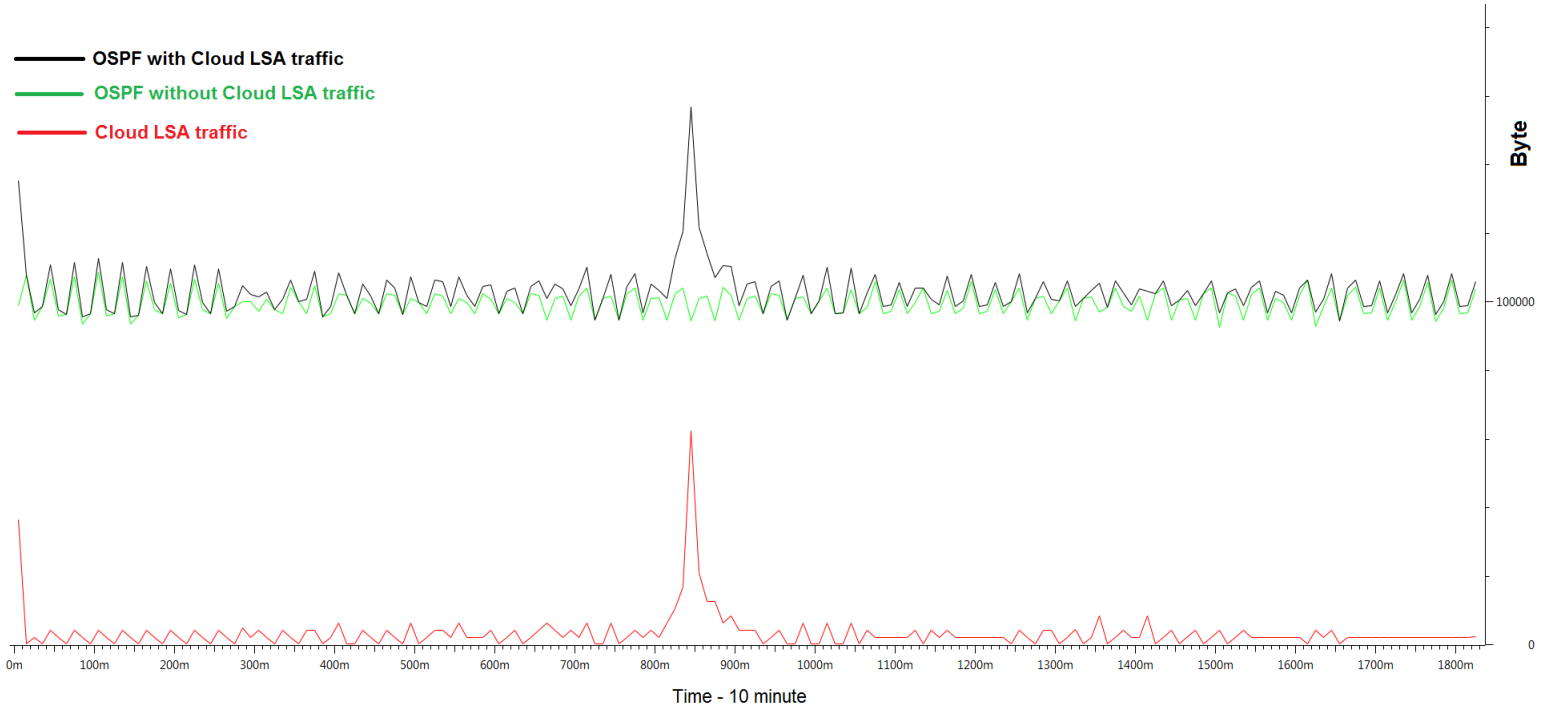

Figure 6.7: Measured OSPF protocol traffic in one test round- bytes per 10 minute. Second scenario: Three embedded data center in network. Protocol traffic for OSPF without Cloud LSA, OSPF with Cloud LSA, and the Cloud LSA traffic

The green line in the above graph shows the traffic due to the fundamental functionality of OSPF protocol. For more information about OSPF's main messages, see section 2.3. The red line shows the traffic due to the Cloud LSAs in the simulated OSPF network. More specifically this red line depicts the extra traffic that the solution adds to the underlying OSPF traffic (shown in green). Finally, the black line depicts the overall OSPF traffic, basic OSPF traffic plus Cloud LSAs traffic, sent through the OSPF network.

The traffic patterns (Figure 6.7, and Figures in Appendix J) show one peak around midday. As discussed in the section 4.1.2 and Appendix B, at midday, the simulated data center resources are at their lowest state. In this condition, based on the relative threshold update policy (i.e. OSPF-Sender module policy in test scenarios), the extended OSPF router sends more frequent Cloud LSUs when the resources state are low (see section 5.4.2). Figure 6.8 shows how the LSUs due to changes in the data center's resources vary during one day.

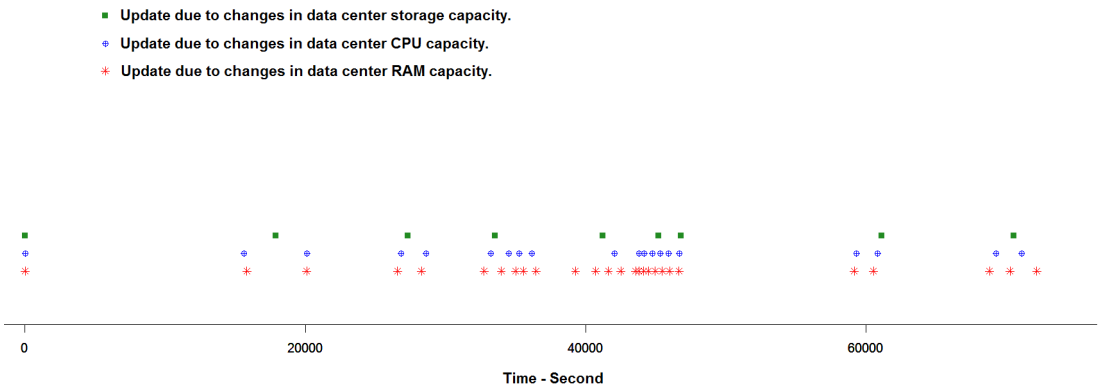

Figure 6.8: The distribution of Cloud LSUs during one day for a proposed data center model when a relative threshold update policy with threshold value 20% is selected.

6.3 Discussion

The statistics shows the additional traffic due to Cloud LSA has a relation with the number of data centers in Cloud network as expected. Adding more data center to Cloud results in more Cloud LSAs, and as a consequence more protocol overhead. Also, the traffic that each sender injects in to the network is differ from the traffic

that receiver nodes see, based on OSPF LSA N-squered problem.

Additionally, the statistics shows in a first scenario 0.590±0.430% extra traffic, in a second scenario 1.780±0.533% extra traffic, and in a third scenario $3.419 \pm 0.565\%$ extra traffic are loaded into OSPF network due to Cloud OSPF LSAs in this particular network embedded cloud. Moreover, the analysis showed that the traffic patterns are biased based on network topology, data center model, and update policy. As a result, by changing the data center model, data center capacity, and update policy the traffic patterns due to Cloud LSAs and test results may changes and these results are valid for our particular data center.

Based on the OSPF N-squared problem, in a OSPF network with N OSPF nodes, sending *one* Cloud LSA can produce in order of N^2 redundant messages (particularly in our test cases sending one cloud LSA produces 13 redundant Cloud LSAs in a network). As a result, with high value "N" the proposed solution faces a scalability issue. To ignore unnecessary messages forwarding in an OSPF network we propose that each router do a simple analysis on each Cloud LSA (e.g., calculate the hash value of Cloud LSA information or look in to the Cloud LSA information). This analysis enable the OSPF routers in a network to recognize whether they saw that LSA before or not. As a result, the OSPF router can discard forwarding the redundant LSAs to other interfaces if it is not required. In this case, we can reduce the unnecessary traffic in OSPF network but with extra cost on router processor. More research and study are required in this area which remains for future work.

It is worth remembering that the OSPF protocol produces less protocol overhead than OSPF-TE, when the network is stable (see section 2.3). Although our simple test scenarios show that the proposed solution in this work produces less protocol overhead than pure OSPF protocol, we expect the proposed solution produces more protocol overhead than pure OSPF (but less protocol overhead than OSPF-TE), when the network scales up (see section 6.2.2). More research and experiments are required to prove or reject this assumption which remains for future work.

Finally, these test scenarios showed that the solution is feasible and it functions properly in a simulated network embedded cloud.

Chapter 7

Conclusions

This chapter summarizes the general conclusions from this thesis project. In addition, some possible future work is proposed for improvements, extensions, and complements to this work. Finally, this chapter ends with some relevant required reflections.

7.1 Conclusion

Network embedded cloud systems are developing and major vendors are trying to gain leadership regarding this new trend. To optimize resource management in a network embedded cloud the Cloud Management System (CMS) requires information not only about a the network conditions, but also details of the resource available at each distributed data center. The initial idea of this project was to understand the current state-of-the-art in network embedded clouds and propose an extension to OSPF-TE that can provide the required virtualization and processing information for a CMS. This project met all the initial goals as described in section 1.4. However, there are rooms to expand upon the performance evaluation which the author did not do, due limited time and the limitation discussed in section 1.6.

In this work, an overview of cloud networks, OSPF, and OSPF-TE was given. Also, in order to help a reader to understand this work, additional topics, including data center resource modeling and link-state update policies, were discussed in the different parts of the thesis. As a result of this master thesis, a prototype for an OSPF-TE extension for network a embedded cloud has been proposed, implemented, and evaluated. In this prototype, a new type of LSA was introduced. This new LSA is referred to as a Cloud LSA. The Cloud LSA is a TE LSA which is extended with a new sub TLV on the Node attribute TLV as was

discussed in section 3.3. The Cloud LSA allows more efficient resource allocation and resource management in a network embedded cloud by providing available resource information (CPU, RAM, Storage, and data center's location). This prototype was implemented using the Quagga OSPF API.

Finally, some basic test scenarios were run to evaluate the performance, suitability, and flexibility of this proposed solution. In order to run these tests, two main decision were made: the data center model was implemented, and an update policy was selected. These two decisions *plus* a test bed topology directly affect the performance of the solution. Therefore, the test results are driven by the selected data center model - which may or may not have any relevance to the rate at which resource changes could occur in actual data centers. The test results shows that this solution produces 7.024 ± 0.529 Bps protocol overhead for a scenarios with six embedded data centers (see section 6.2.3).

However, our simple model for a data center allowed us to evaluate the feasibility and performance of our proposed solution proved in a simulated cloud network. The author calculated the worst case performance of the proposed solution for a mesh connected network embedded cloud with "N" OSPF routers an "D" embedded data center. The results of worst case analysis shows that this solution produces at most $D \times O(N^2) \times 38.4$ Bps with a maximum peak value $D \times O(N^2) \times 192$ *Byte per 5 seconds* protocol overhead. The test and analysis results were given in Chapter 6.

If I had to do this project again, I would have invested more time in identifying the techniques for network resource monitoring and less time in studying different types of cloud models. Also I would have more focused on developing more suitable update policy, i.e., a policy specifically for Cloud resources, and to realize a more realistic model for the data centers.

One of the most important insights to come from this work is the need to understand the new opportunities and requirements based on use cases of a network embedded cloud. Unfortunately, at the time that this thesis was done the concept of network embedded cloud was not fully developed. As a result, I was not able to consider the details of such a system in my solution. This knowledge would help to better plan the design and implementation of solutions for problems in this area. Future work should consider the user and provider requirement in network embedded cloud and then try to find a solution for limitations in resource monitoring and resource management.

7.2 Future work

The solution proposed in this thesis enables available resource monitoring in a network embedded cloud as discussed in section 1.3. There are plenty of opportunities for future research and many improvements could be made. Some of the suggested future work will be addressed below.

There are alternative solutions for resource monitoring in network embedded clouds such as other pull base or push base link-state update protocols. All of the possible solutions were not analyzed and thoroughly studied due to limited time. As a result, the proposed solution may not be the best solution for resource monitoring in a network embedded cloud. More research on other solutions is required to compare the currently proposed solution, i.e., the extended OSPF-TE, with.

Moreover, the data center resource modeling and update policy algorithm were two key factors in our evaluation. While a relative threshold change based update algorithm was used in this work, in the future an advanced update policy specifically for cloud resources should be proposed. Additionally, more information about a data center's resource utilization would help by providing more reliable data which could be used to evaluate different solutions for resource monitoring.

Another compelling issue concerns what is the best way to send the data center resource information with regard to the form of new LSUs. Is that better have a different LSU for each cloud resource parameter or provide all the resource parameter in one LSU? To be able to answer this question and achieve optimized LSUs, more analysis and research are required concerning the Cloud LSA format.

Furthermore, in this thesis project tests was run with three basic scenarios. To collect more reliable data, more advanced scenarios are required. Also, additional work is needed to propose a cloud TLV to the Internet Engineering Task Force (IETF) and to make it a standard.

One of the greatest future steps for this work would be to integrate the solution with commercial routing suite, OpenStack [86], or commercial Network Management System. Also, the current prototype does not support multiple OSPF areas nor IPv6. Finally, the network topology (including links and data centers) should be viewable via in a web interface.

7.3 Required reflections

Working on this project, I gained knowledge about different types of cloud networks and current research questions in distributed and network embedded cloud. Also, I gained knowledge about link-state protocols specifically OSPF and OSPF-TE routing protocol. Moreover, learning and improving my skills in programming languages including C, C++, Python, and Shell Scripting was a great achievement for me. Furthermore, I worked with Linux and various tools and softwares in this project including the Virtual box, Quagga router, and R, which enriched my knowledge and experiences. Finally, it was a great experience and training to solve an industrial problem independently starting with a literature study and continuing with design, implementation, and evaluation.

This project proposed a solution for resource monitoring for a network embedded cloud. As this new cloud paradigm emerges, a network embedded cloud provider can use this proposed solution to monitor their embedded data center resources and network conditions in a single package. Using the information provided by the extended OSPF-TE, a network embedded cloud provider can realized optimal resource management. Also, the end user can take advantage of better services from network embedded cloud providers.

The solution proposed in this work for monitoring distributed data centers in a network embedded cloud has some privacy issues. When the OSPF traffic passes across links that others have access to, the attackers could reveal details of the operator's resources, and resource usage of applications running on the network embedded cloud.

Additionally, there are some social and ethical issues with regard to the carrier providing both communication and processing services. In this model of cloud network, the operator can not only monitor the user traffic, but the operator now actually has access to and is processing the user data. There are some questions such as: should user and/or society trust operators with this information?

In some settings, there are also social issues with regard to potential monopolistic behavior by such a network embedded cloud operator who now not only provides connectivity, processes the user data, but is also probably storing a copy of user data, which could be a very dangerous security situation.

Bibliography

[1] B. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," in *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, Aug. 2009, pp. 44–51.

[2] K. Rafique, A. Tareen, M. Saeed, J. Wu, and S. Qureshi, "Cloud computing economics opportunities and challenges," in *Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on*, oct. 2011, pp. 401–406.

[3] Heavy Reading Service Provider IT Insider, "Big Vendors Race to Establish Carrier Cloud," vol. 8, NO. 1, March 2012.

[4] "Amazon Web Services, Cloud Computing: Compute, Storage, Database," Accessed Oct 4, 2012. [Online]. Available: http://aws.amazon.com/

[5] "Windows Azure: Microsoft's Cloud Platform | Cloud Hosting | Cloud Services," Accessed Oct 4, 2012. [Online]. Available: http://www.windowsazure.com

[6] "Google App Engine Cloud Platform," Accessed Oct 4, 2012. [Online]. Available: https://cloud.google.com/products/

[7] J. Ge, B. Zhang, and Y. Fang, "Research on the resource monitoring model under cloud computing environment," in *Proceedings of the 2010 international conference on Web information systems and mining*, ser. WISM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 111–118.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157 (Historic), Internet Engineering Task Force, May 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

[9] K. Hutton, J. Tiso, M. Schofield, and D. Teare, *Designing Cisco Network Service Architectures (ARCH): (Ccdp Arch 642-874)*, ser. Foundation

Learning Guides. Cisco Press, 2011, ch. 6.Designing Scalable OSPF Design.

[10] Y. Yan, H. Wessing, M. Berger, and L. Dittmann, "Prioritized OSPF-TE Mechanism for Multimedia Applications in MPLS Networks," *Proc. of G/MPLS Networks 2006*, 2006.

[11] K. Shuaib and F. Sallabi, "Extending OSPF for large scale MPLS networks," in *Advances in Wired and Wireless Communication, 2005 IEEE/Sarnoff Symposium on*, 2005, pp. 13–16.

[12] Marben Company, "OSPF-TE, the most widely used protocol in Traffic-Engineered networks," Accessed May 2013. [Online]. Available: http://www.marben-products.com/marben/ospf-te.html

[13] "Quagga Software Routing Suite," Accessed Jun 2012. [Online]. Available: http://www.nongnu.org/quagga/

[14] D. Gelperin, "Exploring agile," in *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral*, ser. APOS '08, 2008, pp. 1–3.

[15] F. Van Der Molen, *Get Ready for Cloud Computing*, ser. Van Haren Series. Van Haren Publishing, 2010.

[16] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008.

[17] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Information Technology Laboratory, Tech. Rep., September 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[18] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A Framework for Native Multi-Tenancy Application Development and Management," in *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*, july 2007, pp. 551–558.

[19] M. Zhou, R. Zhang, D. Zeng, and W. Qian, "Services in the Cloud Computing era: A survey," in *Universal Communication Symposium (IUCS), 2010 4th International*, Oct. 2010, pp. 40–46.

[20] K. Church, A. Greenberg, and J. Hamilton, "On Delivering Embarrassingly Distributed Cloud Services," in *HotNets*, 2008. [Online]. Available: http://conferences.sigcomm.org/hotnets/2008/papers/10.pdf

[21] P. Endo, A. de Almeida Palhares, N. Pereira, G. Goncalves, D. Sadok, J. Kelner, B. Melander, and J. Mangs, "Resource allocation for distributed cloud: concepts and research challenges," *Network, IEEE*, vol. 25, no. 4, pp. 42–46, July-August 2011.

[22] "Empty Promises and Tough Luck: Yankee Group Exposes the Cloud's Fine Print," Boston, MA Apr 21, 2010. [Online]. Available: http://www.yankeegroup.com/about_us/press_releases/2010-04-21.html

[23] "Carrier-centric, Carrier-grade and Founded on IT and Network Innovation," 2010, Accessed: 01/09/2012. [Online]. Available: http://www.slideshare.net/NECIndia/nec-carrier-cloud

[24] "Ericsson announces Network-enabled Cloud concept at Mobile World Congress - TelecomLead.com: Telecom News on Mobile, Wireless Infrastructure, Enterprise Networking, Smartphone, Tablet, LTE, VAS." [Online]. Available: http://www.telecomlead.com/telecom-equipment/ericsson-announces-network-enabled-cloud-concept-at-mobile-world-congress/

[25] "IBM: Cloud strategies for virtualization of telecom infrastructure." [Online]. Available: http://www.telecomlead.com/cloud/ibm-cloud-strategies-for-virtualization-of-telecom-infrastructure-77130/

[26] Alcatel-Lucent and HP, "Cloud Ready Service Infrastructure for Communications Service Providers," STRATEGIC WHITE PAPER, 2011. [Online]. Available: http://www.telecoms.com/wp-content/blogs.dir/1/files/2011/10/HP_ALU_Cloud_WhitePaper110613-3.pdf

[27] Kontron, "Cloud Evolution: The Carrier Cloud," White Paper. [Online]. Available: http://emea.kontron.com/_etc/scripts/application/getcollateral.php?file=kontroncarriercloudwp.pdf

[28] B. Sosinsky, *Cloud Computing Bible*, 1st ed. Wiley Publishing, 2011.

[29] C. Pawar and R. Wagh, "A review of resource allocation policies in cloud computing," *World Journal of Science and Technology*, vol. 2, no. 3, 2012. [Online]. Available: http://worldjournalofscience.com/index.php/wjst/article/view/13190/6680

[30] J. Kurose and K. Ross, "Routing algorithms," in *Computer Networking: A Top-Down Approach (Fifth Edition)*. Addison-Wesley, 2010.

[31] C. Hedrick, "Routing Information Protocol," *Internet Request for Comments*, vol. RFC 1058 (Historic), Jun. 1988, updated by RFCs 1388, 1723. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1058.txt

[32] C. L. Rutgers, "An introduction to IGRP," The State University of New Jersey, Center for Computers and Information Services, Laboratory for Computer Science Research, Tech. Rep., Aug. 1991.

[33] J. Moy, "OSPF Version 2," *Internet Request for Comments*, vol. RFC 2328 (Standard), Apr. 1998, updated by RFCs 5709, 6549. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2328.txt

[34] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," *Internet Request for Comments*, vol. RFC 1142 (Informational), Feb. 1990. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1142.txt

[35] E. W. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[36] K. Kompella and B. Fenner, "IANA Considerations for OSPF," *Internet Request for Comments*, vol. RFC 4940 (Best Current Practice), Jul. 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4940.txt

[37] J. Moy, "Multicast Extensions to OSPF," RFC 1584 (Historic), Internet Engineering Task Force, Mar. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1584.txt

[38] P. Murphy, "The OSPF Not-So-Stubby Area (NSSA) Option," RFC 3101 (Proposed Standard), Internet Engineering Task Force, Jan. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3101.txt

[39] R. Coltun, "The OSPF Opaque LSA Option," RFC 2370 (Proposed Standard), Internet Engineering Task Force, Jul. 1998, obsoleted by RFC 5250, updated by RFC 3630. [Online]. Available: http://www.ietf.org/rfc/rfc2370.txt

[40] L. Berger, I. Bryskin, A. Zinin, and R. Coltun, "The OSPF Opaque LSA Option," *Internet Request for Comments*, vol. RFC 5250 (Proposed Standard), Jul. 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5250.txt

[41] John Moy, "Open shortest path first (OSPF) opaque link-state advertisements (LSA) option types," Jan. 2009. [Online]. Available: http://www.iana.org/assignments/ospf-opaque-types/ospf-opaque-types.xml

[42] D. Katz, K. Kompella, and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2," *Internet Request for Comments*, vol. RFC 3630 (Proposed Standard), Sep. 2003, updated by RFCs 4203, 5786. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3630.txt

[43] J. Moy, P. Pillay-Esnault, and A. Lindem, "Graceful OSPF Restart," RFC 3623 (Proposed Standard), Internet Engineering Task Force, Nov. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3623.txt

[44] A. Lindem, N. Shen, J. Vasseur, R. Aggarwal, and S. Shaffer, "Extensions to OSPF for Advertising Optional Router Capabilities," RFC 4970 (Proposed Standard), Internet Engineering Task Force, Jul. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4970.txt

[45] I. Bryskin and L. Berger, "OSPF-Based Layer 1 VPN Auto-Discovery," RFC 5252 (Proposed Standard), Internet Engineering Task Force, Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5252.txt

[46] M. Chen, R. Zhang, and X. Duan, "OSPF Extensions in Support of Inter-Autonomous System (AS) MPLS and GMPLS Traffic Engineering," RFC 5392 (Proposed Standard), Internet Engineering Task Force, Jan. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5392.txt

[47] O. Younis and S. Fahmy, "Constraint-Based Routing in the Internet: Basic Principles and Recent Research," *IEEE Communications Surveys and Tutorials*, vol. 5, pp. 2–13, 2003.

[48] A. Karaman, "Constraint-Based Routing in Traffic Engineering," in *Computer Networks, 2006 International Symposium on*, 0-0 2006, pp. 1–6.

[49] K. Ishiguro, V. Manral, A. Davey, and A. Lindem, "Traffic Engineering Extensions to OSPF Version 3," RFC 5329 (Proposed Standard), Internet Engineering Task Force, Sep. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5329.txt

[50] K. Kompella and Y. Rekhter, "OSPF Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)," RFC 4203 (Proposed Standard), Internet Engineering Task Force, Oct. 2005, updated by RFCs 6001, 6002. [Online]. Available: http://www.ietf.org/rfc/rfc4203.txt

[51] R. Aggarwal and K. Kompella, "Advertising a Router's Local Addresses in OSPF Traffic Engineering (TE) Extensions," RFC 5786 (Proposed Standard), Internet Engineering Task Force, Mar. 2010, updated by RFC 6827. [Online]. Available: http://www.ietf.org/rfc/rfc5786.txt

[52] A. V. Aho and D. Lee, "Hierarchical networks and the LSA N-squared problem in OSPF routing," in *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE*, vol. 1, 2000, pp. 397–404 vol.1.

[53] A. Shaikh, J. Rexford, and K. G. Shin, "Evaluating the impact of stale link state on quality-of-service (QoS) routing," *IEEE/ACM Trans. Netw.*, vol. 9, no. 2, pp. 162–176, Apr. 2001. [Online]. Available: http://dx.doi.org/10.1109/90.917073

[54] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, and S. Tripathi, "Intradomain QoS routing in IP networks: a feasibility and cost/benefit analysis," *Network, IEEE*, vol. 13, no. 5, pp. 42–54, 1999.

[55] X. Yuan, W. Zheng, and S. Ding, "A comparative study of QoS routing schemes that tolerate imprecise state information," in *Computer Communications and Networks, 2002. Proceedings. Eleventh International Conference on*, 2002, pp. 230–235.

[56] T. il Kim, J.-J. Yoo, H.-W. Jung, H.-H. Le, M. Y. Chung, and S.-I. Jin, "Adaptive threshold-based link status update mechanism," in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, vol. 1, feb. 2006, pp. 888–891.

[57] A. Basu and J. Riecke, "Stability issues in OSPF routing," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 225–236, Aug. 2001.

[58] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi, "Improving QoS Routing Performance Under Inaccurate Link State Information," pp. 7–11, Proceedings of the 16th International Teletraffic Congress.

[59] M. Handley, O. Hodson, and E. Kohler, "Xorp: An open platform for network research," in *ACM SIGCOMM Computer Communication Review*, 2002, pp. 53–57.

[60] "The BIRD Internet Routing Daemon Project," Accessed Aug 2012. [Online]. Available: http://bird.network.cz/

[61] R. Keller, "Dissemination of Application-Specific Information Using the OSPF Routing Protocol," TIK Report Nr. 181, TIK, ETH Zurich, November 2003.

[62] G. Goncalves, M. Santos, G. Charamba, P. Endo, D. Sadok, J. Kelner, B. Melander, and J.-E. Mangs, "D-CRAS: Distributed cloud resource allocation system," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, april 2012, pp. 659–662.

[63] Q. Zhang, Q. Zhu, M. Zhani, and R. Boutaba, "Dynamic Service Placement in Geographically Distributed Clouds," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, june 2012, pp. 526–535.

[64] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 963–971.

[65] V. Visockas, "Comparing expected and real-time spotify service topology," Master's thesis, KTH, Communication Systems, TRITA-ICT-EX-2012:63, May 2012,http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-96352.

[66] M. Dhingra, J. Lakshmi, and S. K. Nandy, "Resource Usage Monitoring in Clouds," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, ser. GRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 184–191. [Online]. Available: http://dx.doi.org/10.1109/Grid.2012.10

[67] H. Huang and L. Wang, "P amp;P: A Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 260–267.

[68] R. Keller and B. Plattner, "Self-Configuring Active Services for Programmable Networks," in *IWAN*, 2003, pp. 137–150.

[69] L. Zuliani, M. Savasini, G. Pavani, R. Pasquini, F. Verdi, and M. Magalhaes, "An implementation of an OSPF-TE to support GMPLS-controlled all-optical WDM networks," in *Telecommunications Symposium, 2006 International*, 2006, pp. 300–305.

[70] M. Zhao, H. Zhu, V. Li, and Z. Ma, "A stability-based link state updating mechanism for QoS routing," in *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, vol. 1, may 2005, pp. 33–37 Vol. 1.

[71] A. Ariza, E. Casiliari, and F. Sandoval, "Strategies for updating link states in QoS routers," *Electronics Letters*, vol. 36, no. 20, pp. 1749–1750, sep 2000.

[72] T. il Kim, J.-J. Yoo, H.-W. Jung, H.-H. Lee, M. Y. Chung, and S.-I. Jin, "Link State Update Algorithm considering Traffic Variation," in *Advanced Communication Technology, The 9th International Conference on*, vol. 2, feb. 2007, pp. 1271–1274.

[73] M. Noordermeer, "Implementing link-state update policies for quality of service routing," Master's thesis, TUDelft, Electrical Engineering, Mathematics and Computer Science, Telecommunications, 2011-11-10. [Online]. Available: http://repository.tudelft.nl/view/ir/uuid:665d21a2-2289-4ab4-b709-8fc58de0ea95/

[74] "Amazon's CloudWatch," Accessed April 2013. [Online]. Available: http://aws.amazon.com/cloudwatch/

[75] "CloudClimate, Cloud Hosting and Cloud Storage Performance Dashboard," Accessed April 2013. [Online]. Available: http://www.cloudclimate.com/

[76] "CloudKick," Accessed April 2013. [Online]. Available: https://www.cloudkick.com/

[77] "SQLite," Accessed Jun 2013. [Online]. Available: http://www.sqlite.org/

[78] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi, "Quality of service based routing: a performance perspective," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 17–28, Oct. 1998. [Online]. Available: http://doi.acm.org/10.1145/285243.285251

[79] ImageStream Internet Solutions, Inc., "The OSPF Routing Protocol." [Online]. Available: http://support.imagestream.com/OSPF.html

[80] Data Network Resource, "OSPF routing protocol, dijkstra algorithm, OSPF stub area." [Online]. Available: http://www.rhyshaden.com/ospf.htm

[81] Intel, "Intel® Hyper-Threading Technology *(Intel HT Technology)*." [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

[82] J. Watson, "Virtualbox: bits and bytes masquerading as machines," *Linux J.*, vol. 2008, no. 166, Feb. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1344209.1344210

[83] A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce, *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.

[84] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[85] "Service Name and Transport Protocol Port Number Registry," Accessed May 2013. [Online]. Available: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml

[86] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Article: Openstack: Toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, October 2012, published by Foundation of Computer Science, New York, USA.

Appendix A

Embedded data center module source code

The embedded data center's module is written with the C programming language. This module simulates a data center with a predefined capacity. To describe the data center's capacity a data structure *"ComputeResource"* is used as below;

```
struct ComputeResource {
        int RAM; //GB
        int CPU; //GHz
        int Storage; //GB
};
```

The simulated data center is described by following predefined values as a data center's capacity.

```
// Total Resources; Ram(GB), CPU(GHz), Storage (GB)
const struct ComputeResource TotalResource = {12000, 176000, 6600000};

// Usable capacity of data center
const struct ComputeResource Capacity = {12000*0.85 , 176000*0.85, 6600000*0.85};

// Available usable capacity
struct ComputeResource CurrentResource = {12000*0.85 , 176000*0.855, 6600000*0.85}
```

The *"TotalResource"* data structure represents the total available resources of data center. All available capacity of data center is not utilizable by customers. In this data center design, the policy is not to announce 15% of total resources to customer and keep it as a surplus. This can occur due to maintenance or

avoid bad decision by Resource Management System, see chapter 5. As a result, the data center's utilization never exceeds more than almost 85%. The *"Capacity"* data structure represents the functional capacity of data center. Finally, the *"CurrentResource"* represents the current functional capacity of a data center. The CurrentResource values changes when the data center serves requests.

The data center module supports 14 different type of instances. These instances are close upon amazon EC2 instances [4]. Each instance provides a predictable amount of dedicated compute capacity. The instances defined by fixed values for CPU (GHz), RAM (GB), and storage (GB) as follows;

```
// Instance 1
struct ComputeResource M1SmallInstance = {.RAM = 1, .CPU = 1, .Storage = 160};
// Instance 2
struct ComputeResource M1MediumInstance = {3, 2, 410};
// Instance 3
struct ComputeResource M1LargeInstance = {7, 4, 750};
// Instance 4
struct ComputeResource M1ExLargeInstance = {15, 8, 1500};

// Instance 5
struct ComputeResource M3ExLargeInstance = {15, 13, 0 };
// Instance 6
struct ComputeResource M3DoubleExLargeInstance = {30, 26, 0};

// Instance 7
struct ComputeResource MicroInstance = {0, 2, 0};

// Instance 8
struct ComputeResource HighMemExLargeInstance = {17, 6, 420};
// Instance 9
struct ComputeResource HighMem2ExLargeInstance = {34, 13, 850};
// Instance 10
struct ComputeResource HighMem4ExLargeInstance = {68, 26, 1700};
// Instance 11
struct ComputeResource HighCPUMediumInstance = {2, 5, 350};
// Instance 12
struct ComputeResource HighCPUExLargeInstance = {7, 20, 1600};

// Instance 13
struct ComputeResource Clus4ExLargeInstance = {23, 33, 1600};
// Instance 14
```

```
struct ComputeResource Clus8ExLargeInstance = {60, 88, 3300};
```

The application uses the *"rand ()"* function to generate the random number. To prevent generating of the same sequence in each round, the *"srand (x)"* function is used. This function sets the seed of the random number generator algorithm. The application uses the computer's internal clock to control the choice of the seed for the *"srand ()"* function. Since time is continually changing, the *"rand ()"* function generates a different sequence of random values in each run.

```
srand(time(NULL));
```

The program uses biased probability to choose the job type, see section 4.1. In another words, in each request interval the program generates a random number between 1 and 100. Then, based on the hour for this data center utilization and a generated random value the application chooses one of four defined job types. The Following show this operation:

```
jobbiased = (rand() % 100) +1 ; // Random value between 1 and 100

switch (dayhour) {
 // first working hours of data center , 12:00 A.M - 12:59 A.M
 case 1:
  ++hourcounter;
  if (jobbiased <= 25) // 25%  of allocation
   job = 0;
  else if (jobbiased <= 50 ) // 25%  of de-allocation
   job = 1;
  else if (jobbiased <= 75 ) //  25%  upgrading
   job = 2;
  else if (jobbiased <= 100 ) // 25%   downgrading
   job = 3;
  // Update the information in Resource data base
  MeanCpuUti = AvearageUTICalculator (MeanCPUUTI, CPUUTI(), hourcounter);
  MeanRamUti = AvearageUTICalculator (MeanRAMUTI, RAMUTI(), hourcounter);
  MeanStorageUti =
  AvearageUtiCalculator (MeanStorageUti, StorageUti(), hourcounter);
  if (TimeIsOver()) ) {
   // the arrival rate also changes in each working hours of data center
   NextHour(hour1) // set the time to next hours
   }
```

```
        ....
        ....
        ....

 // 11 A.M - 11:59 A.M
 case 11:
  ++hourcounter;
  if (jobbiased <= 28) // 50% of allocation
   job = 0;
  else if (jobbiased <= 51 ) //30% of de-allocation
   job = 1;
  else if (jobbiased <= 79 ) // 15%  upgrading
   job = 2;
  else if (jobbiased <= 100 ) // 5% downgrading
   job = 3;

        ....
        ....
        ....
```

when the application makes a decision about job type, then it is a time for completing that action. The following function is responsible for a creation of a virtual machine(s) in a a data center.

```
struct ComputeResource CreateVM () {
 int CreatJob, NumberofVMs;
 int i = 0;
 // Define number of VM
 NumberofVMs = (rand() % 5) + 1;
 // do we have enough available capacity in data center?
 if ( CreationIsAllowd() ) {
  for (i = 0; i < NumberofVMs; i++){
   // chose Instances type randomly
   CreatJob = rand() % numberofjobsVMcreation ;
   Allresource(CreatJob);
           }
   }
   else{
        AllPolicy = 0; // Allocation not allowed any more
    }
   }
```

The Resource DB contains information about a data center's resources. The application continuously updates the Resource DB information after execution of

each job type. The Resource DB is build up on SQLite. The application uses following function to update the resource data base.

```
int wrsqlite (int cpu, int ram, int storage){
 int retval;
 char string[150];
 sqlite3 *conn;
 //Name of data base
 char *database = "resource.db";

 /* Conection to database
  pass a pointer to the pointer to sqlite3, in short sqlite3**
 */
 retval = sqlite3_open(database,&conn);

 // error handeling ---If connection failed, conn returns NULL
 if(retval){
  return 0;
 }

 // Set the string
 memset(string, 0, sizeof(string));
 sprintf(string, "UPDATE resource SET cpu = %d, ram = %d, storage = %d , location = 'S'", cpu

 // execute the string
 retval = sqlite3_exec(conn,string,0,0,0);
 // Execute the query for initialize the values

 // Close the handle to free memory
 sqlite3_close(conn);
 return 1;
}
```

Appendix B

Embedded data center model results

In this part the mean utilization for CPU, RAM, and storage for a simulated data center and its upper and lower bounds of a 95% CI for a simulated data center over 24 hours, which was explained in section 3.2.1.2 and Section 4.1, are shown in Figure B.1, Figure B.2, and Figure B.3.

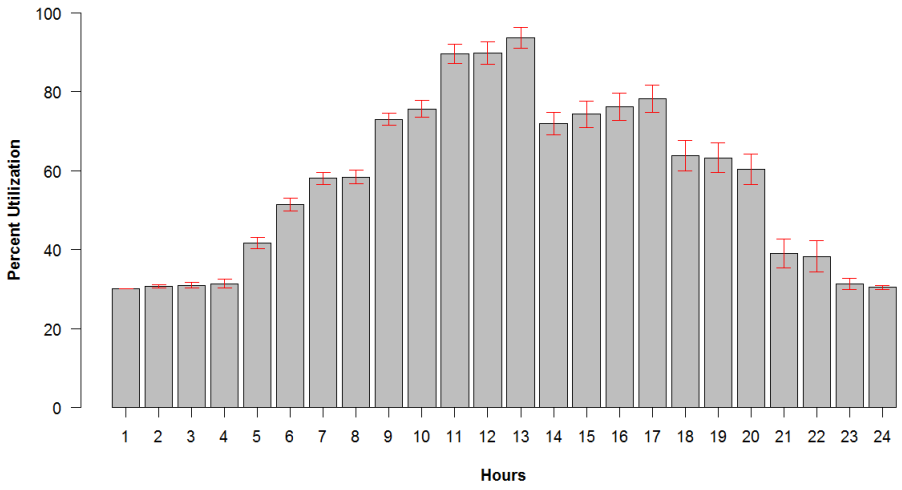

Figure B.1: Simulated data center's mean *CPU* utilization per hour.

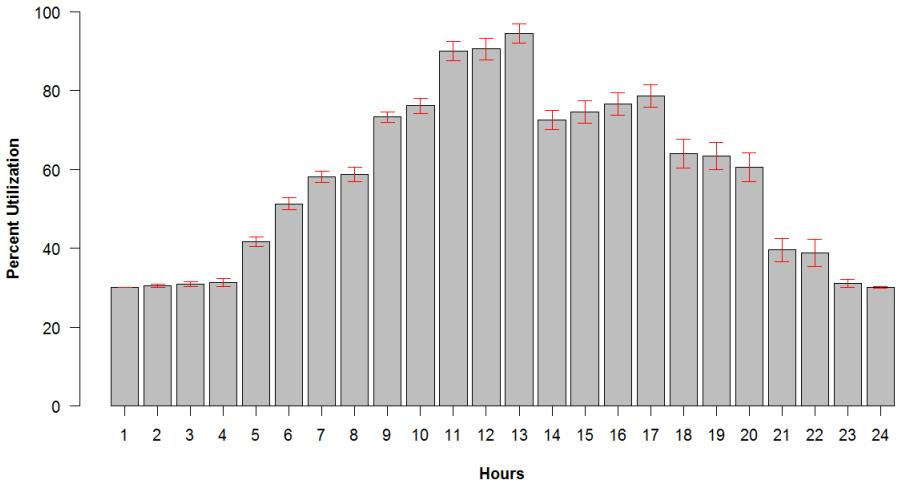

Figure B.2: Simulated data center's mean *RAM* utilization per hour.

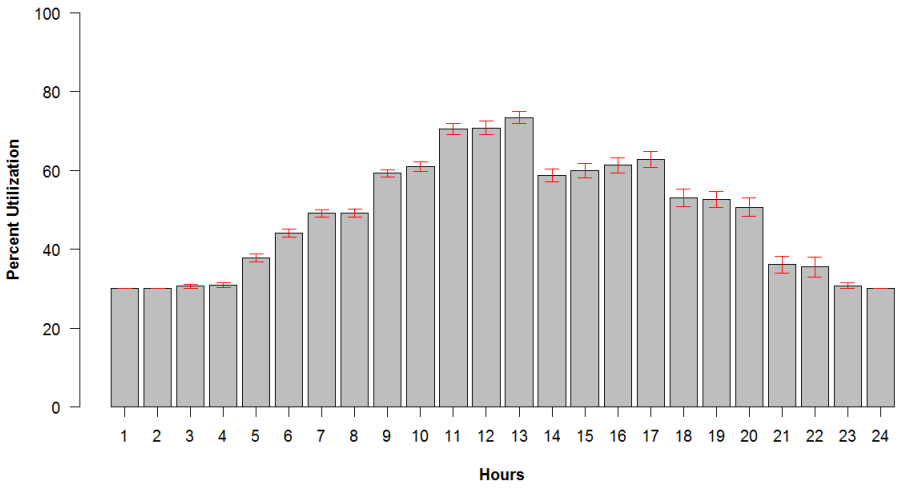

Figure B.3: Simulated data center's mean *storage* utilization per hour.

Figure B.4, Figure B.5, and Figure B.6 represent the CPU, RAM, and storage utilized of a simulated data center per second.

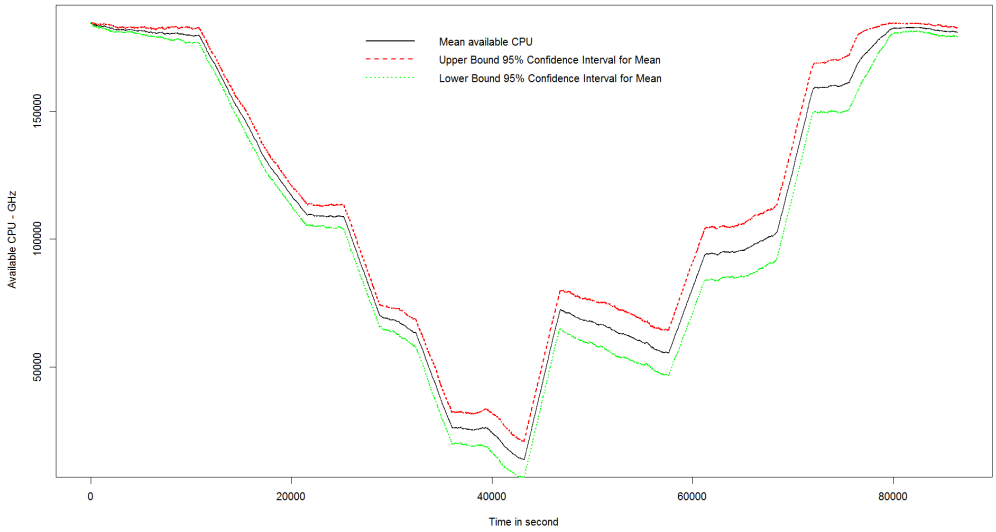

Figure B.4: Simulated data center *CPU* (GHz) capacity utilized per second.

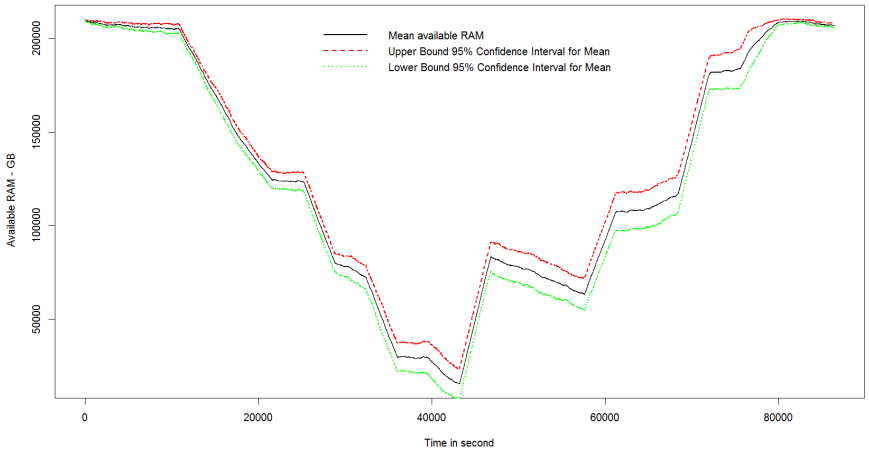

Figure B.5: Simulated data center *RAM* (GB) capacity utilized per second.

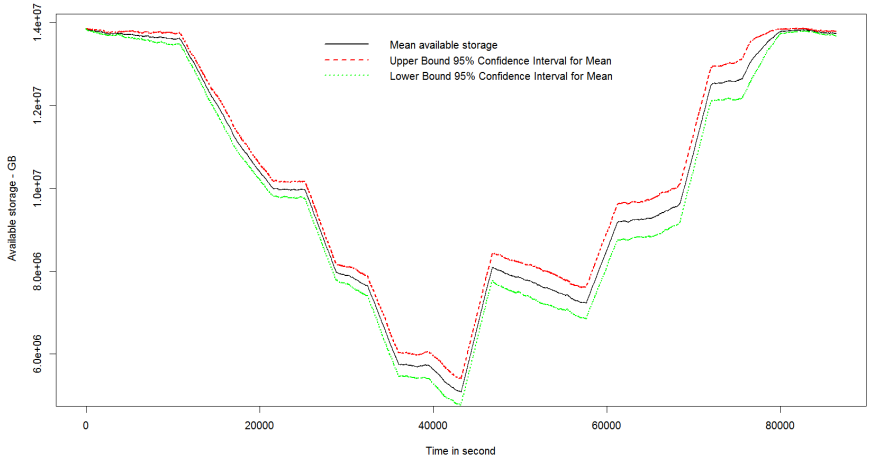

Figure B.6: Simulated data center's *storage* (GB) capacity utilized per second.

Appendix C

Cloud-OSPF-Sender source code

The Cloud-OSPF-Sender module is written in C programming language. User needs to run this module with some parameters to configure the application. If the module does not detect appropriate parameters, it will conclude with the following function.

```
static int usage()
{
  printf("Usage:
  "ospfclient <Target IP> <LSA type> <opaquetype> <opaqueid> <Threshold> <Interval>\n");
  printf("Target IP :"
  "router where API-enabled OSPF daemon is running\n");
  printf("      LSA type  : "
  "either 9, 10, or 11 depending on flooding scope\n");
  printf("      opaquetype: "
  "0-255 (e.g., experimental applications use > 128)\n");
  printf("      opaqueid  : "
  "arbitrary application instance (24 bits)\n");
  printf("      areaid    : "
  "area in IP address format\n");
  printf("      Threshold : "
  "value for LSU policy (1 till 100)\n");
  printf("      Interval  : "
  "Interval for cheeking data center resource DB\n");
  exit(1);
}
```

The Cloud-OSPF-Sender supports concurrency and uses threads for its functionality. Following line of program is used to create master thread. It is worth highlighting

that this module uses Quagga libraries for creation the and handling the threads.

```
master = thread_master_create ();
```

The application uses OSPF-API Built-in function, *"ospf_ apiclient_ connect()"*, to establish TCP connection to OSPF API. The Cloud-OSPF-Sender module calls this function with Target router IP address and NEC-OSPF-TE as a port number. This function will provide two connections between the Cloud-OSPF-Sender and OSPF router. First connection is responsible for synchronous requests/replies to the Quagga API by client. Then, the OSPF router opens a reverse connection for asynchronous messages as a reaction (second connection). The following show some lines of code which are responsible for setting up a connection.

```
//args[1] = Target IP , ASYNCPORT = 5000
oclient = ospf_apiclient_connect (args[1], ASYNCPORT);
if (!oclient){
 printf ("Connecting to OSPF daemon on %s FAILED!\n",
            args[1]);
 exit (1);
}
```

This program uses Resource DB to access to the data center's resource information. To achieve this goal, the *"read_ DC_ data ()"* function is used. The following show this function.

```
int read_DC_data(struct thread *thread)
{
 int rsv;
 //Read data from Resourc DB
 rsv = db_resource_read();

 if (rsv == 0){ // Find new data
  // add lsa_inject_cloud to thread que
  thread_add_timer (master, lsa_inject_cloud, oclient, 0);
  //Reinitialize the read_node_data with Interval
  thread_add_timer (master, read_node_data, oclient, Interval);
  return 0;
 }
 elseif (rsv == -1){ //error on reading DB
  //try reading again
```

```
  thread_add_timer (master, read_node_data, oclient, 0);
  return -1;
 }
 else{// No new data for DC
  //Reinitilaize the read_node_data with Interval
  thread_add_timer (master, read_node_data, oclient, Interval);
  return 1;
 }
}
```

Following function is responsible for reading a data from Resource DB.

```
int db_resource_read ()
{
 int retval;
 char string[150];
 //handler for database connection
 sqlite3 *conn;
 //DB name
 char *database = "resource.db";
 // A prepared statement for fetching tables
 sqlite3_stmt *stmt;
 // number of columns of DB table
 int cols;
 int CPU = 0;
 int RAM = 0;
 int Storage = 0;
 int rsv;
 int i;
 // try to connect to DB
 retval = sqlite3_open(database,&conn);

 // error handeling
 if(retval){
  //conection failed
  return -1;
 }

 // Set the string
 memset(string, 0, sizeof(string));
 // select those all rows from DB
 sprintf(string, "SELECT * from resource");
 retval = sqlite3_prepare_v2(conn,string,-1,&stmt,0);
```

```
if(retval){
 //conecction DB Failed
 DBFetchFaile++;
 //Function for monitoring Data center
 SystemMonitor();
 //SQLite finalization
 sqlite3_finalize(stmt);
 rsv=sqlite3_close(conn);
 return -1;
}

// Read the number of rows fetched
cols = sqlite3_column_count(stmt);
while(1){
 // fetch a row's status
 retval = sqlite3_step(stmt);
 if(retval == SQLITE_ROW){
   for(i=0 ; i<cols;i++){
        // copy the value of column i to val
        const char *val = (const char*)sqlite3_column_text(stmt,i);
        switch(i){
      case 0:
           CPU = atoi(val);
               break;
      case 1:
            RAM = atoi(val);
                 break;
      case 2:
             Storage = atoi(val);
              break;
      case 3:
              location = *val;
              break;
      }
   }
  }
  else if(retval == SQLITE_DONE){
       // All rows finished
      break;
  }
  else{
         // Some error encountered
       sqlite3_finalize(stmt);
```

```
        sqlite3_close(conn);
        return -1;
    }
   }
 // Close the handle to free memory
 sqlite3_finalize(stmt);
 rsv=sqlite3_close(conn);
 //check if data base provide new info or not
 rsv = DC_changelog(CPU,RAM,Storage,location);
 return rsv;
}
```

This function is responsible for applying update policy.

```
int DC_changelog(int CPU,int RAM,int Storage,char location)
{
 float CpuRateChange, RamRateChange, StoRateChange;
 //type casting
 CpuRateChange = abs((cldata->CPU - CPU) ) / (float) abs(  cldata->CPU );
 RamRateChange = abs((cldata->RAM - RAM) ) / (float) abs(  cldata->RAM );
 StoRateChange = abs((cldata->Storage - Storage) ) / (float) abs(  cldata->Storage );

 if (CpuRateChange > Threshold ||
          RamRateChange > Threshold ||
          StoRateChange > Threshold ){
  insert_new_data(CPU,RAM,Storage,location);
  return 0;
 }
 else{
  //NO CHANGE
  return 1;
 }
}
```

Cloud-OSPF-sender uses following function to invoke OSPF router to send LSA.

```
static int
lsa_inject_cloud (struct thread *t)
{
  struct ospf_apiclient *cl;
  struct in_addr ifaddr;
```

```
struct in_addr area_id;
u_char lsa_type;
u_char opaque_type;
u_int32_t opaque_id;
void *opaquedata;
int opaquelen;
int rc;
struct SUB_TLV3 subtlv3;
struct TE_TLV5  tetlv5;
cl = THREAD_ARG (t);

inet_aton (args[5], &ifaddr);
inet_aton (args[6], &area_id);
lsa_type = atoi (args[2]);
opaque_type = atoi (args[3]); // it should be 10
opaque_id = atoi (args[4]);

subtlv3.type = 3;

subtlv3.router_resource.CPU = cldata->CPU;
subtlv3.router_resource.RAM = cldata->RAM;
subtlv3.router_resource.Storage = cldata->Storage;
subtlv3.router_resource.location = cldata->location;

/*  Set the length part of sub TLV of node attribute TLV
 *which is equal to size data that carries in TLV*/
subtlv3.length = sizeof (struct cloud_data);

// Set VALU part of node att TLV
tetlv5.subtlv3 = subtlv3;
tetlv5.type = 5;
/*Set the length of Node Att TLV =
 *length of sub TLV + length of type file + length of length field*/
tetlv5.length = subtlv3.length + sizeof(u_int16_t) + sizeof(u_int16_t);

opaquedata = &tetlv5;

/* Set the length of Opaque data =
length part  TLV  + length of type filed + length of length field*/
opaquelen = tetlv5.length + sizeof(u_int16_t) + sizeof(u_int16_t);

//Invoke OSPF router to send LSA
rc = ospf_apiclient_lsa_originate(cl, ifaddr, area_id,
```

```
                                    lsa_type,
                                    opaque_type, opaque_id,
                                    opaquedata, opaquelen);
  OpaqueUpdateSend++;
  SystemMonitor();
  return 0;
}
```

Following data structures are used for making a Cloud LSA.

```
/* Our Cloud LSAs have the following format. */
struct my_opaque_lsa
{
  /* include common LSA header */
  struct lsa_header hdr;
  /* DC data in format of TE TLV */
  struct TE_TLV5 DC_data;
};

/* OSPF LSA header. */
struct lsa_header
{
  u_int16_t ls_age;
  u_char options;
  u_char type;
  struct in_addr id;
  struct in_addr adv_router;
  u_int32_t ls_seqnum;
  u_int16_t checksum;
  u_int16_t length;
};

// Our Opaque data is in format of TE TLV
struct TE_TLV5
{
        u_int16_t type;
        u_int16_t length;
        struct SUB_TLV3  subtlv3;

};

// We propose sub TLV = 3 for DC resource
struct SUB_TLV3
```

```
{
        u_int16_t type;
        u_int16_t length;
        struct cloud_data DC_resource;

};


// sen DC resources + Location
struct cloud_data
{
         u_int32_t CPU;
         u_int32_t RAM;
         u_int32_t Storage;
         char location;

};
```

Appendix D

Periodic update policy performance analysis

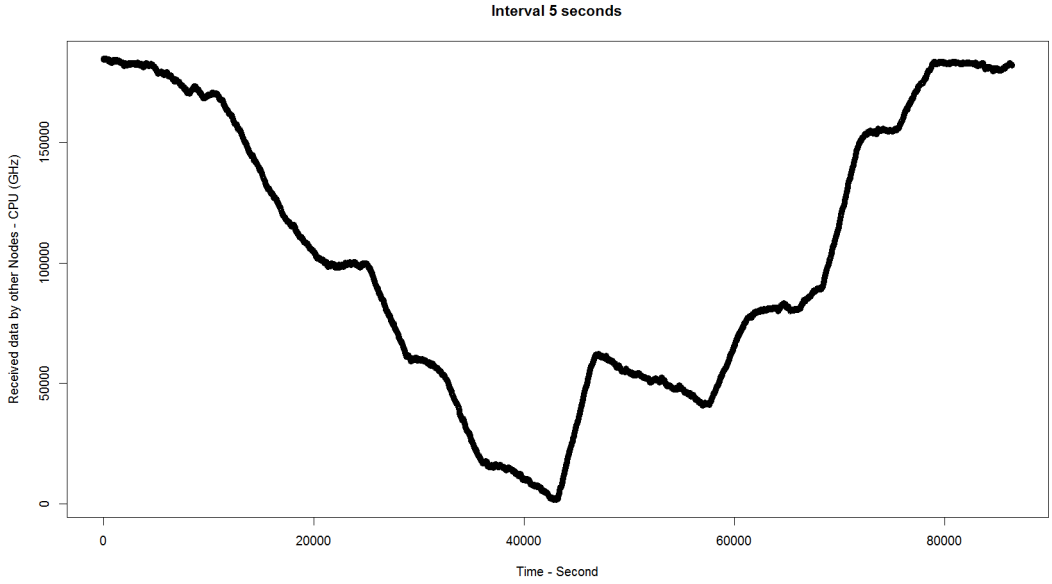

Figure D.1: How a cloud management system views cloud resources with a hold-down timer value 5 seconds, the updates points are plotted a circles.

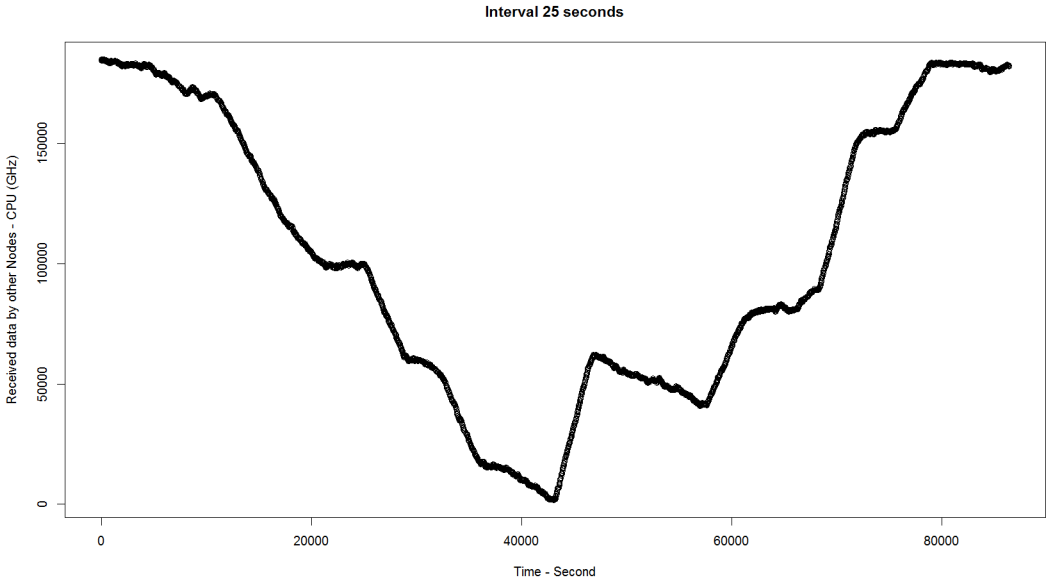

Figure D.2: How a cloud management system views cloud resources with a hold-down timer value 25 seconds, the updates points are plotted a circles.

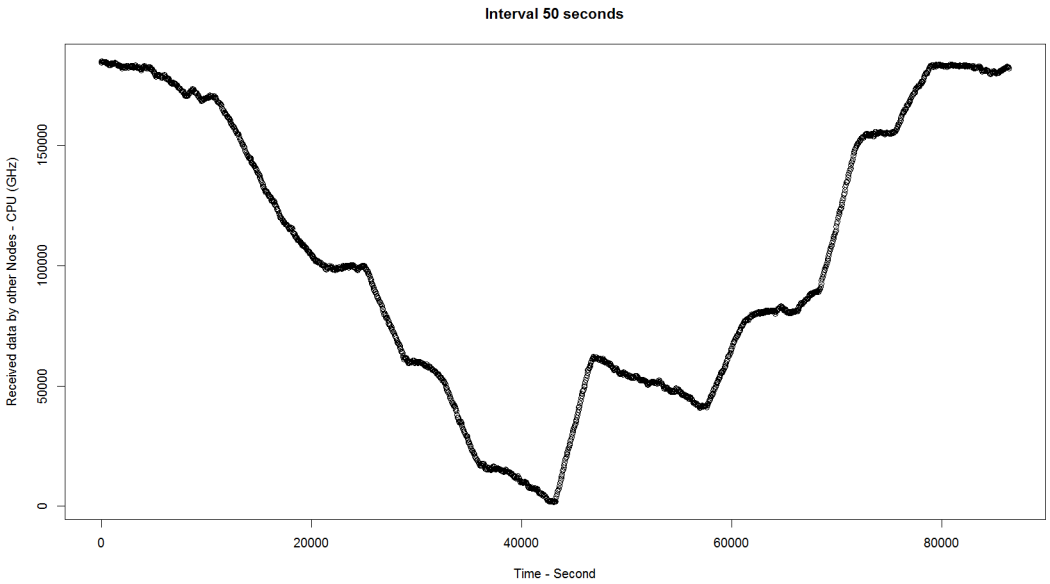

Figure D.3: How a cloud management system views cloud resources with a hold-down timer value 50 seconds, the updates points are plotted a circles.

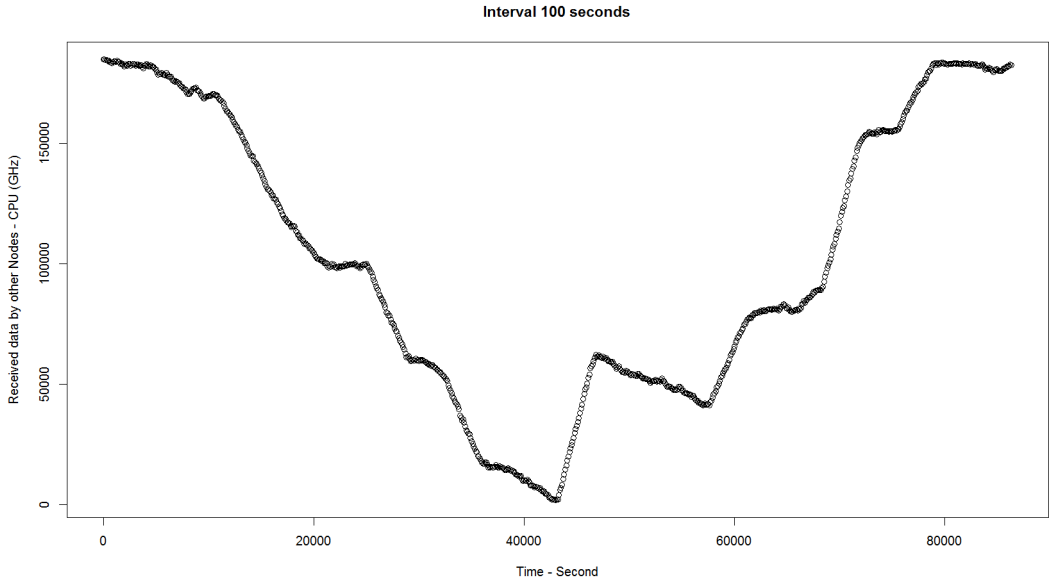

Figure D.4: How a cloud management system views cloud resources with a hold-down timer value 100 seconds, the updates points are plotted a circles.

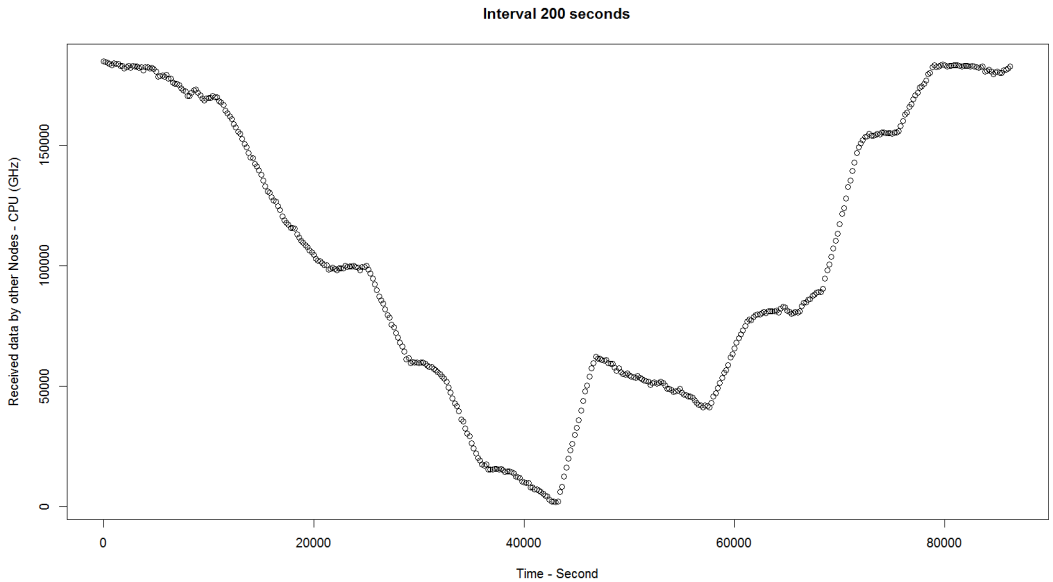

Figure D.5: How a cloud management system views cloud resources with a hold-down timer value 200 seconds, the updates points are plotted a circles.

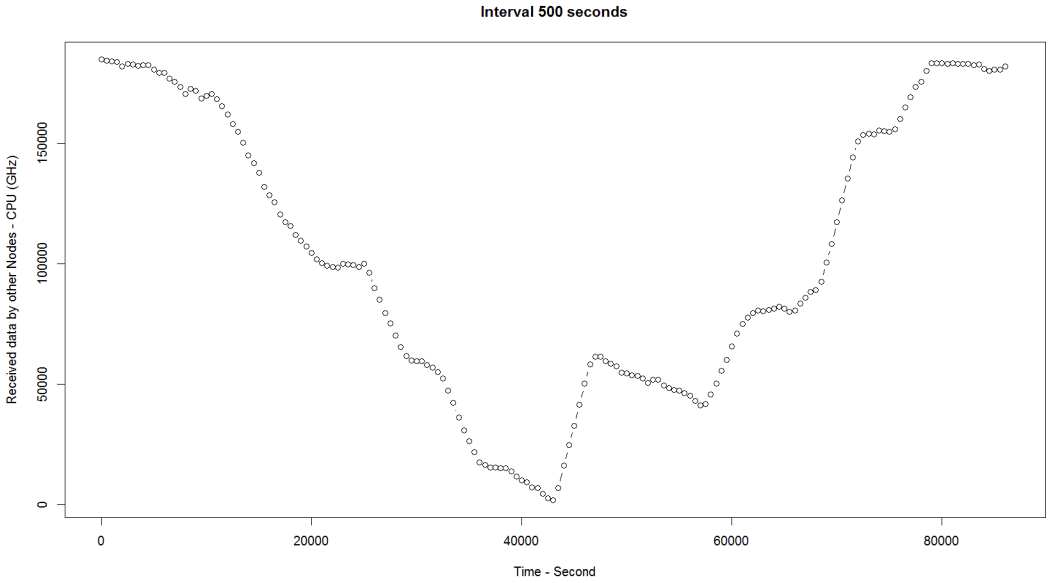

Figure D.6: How a cloud management system views cloud resources with a hold-down timer value 500 seconds, the updates points are plotted a circles.

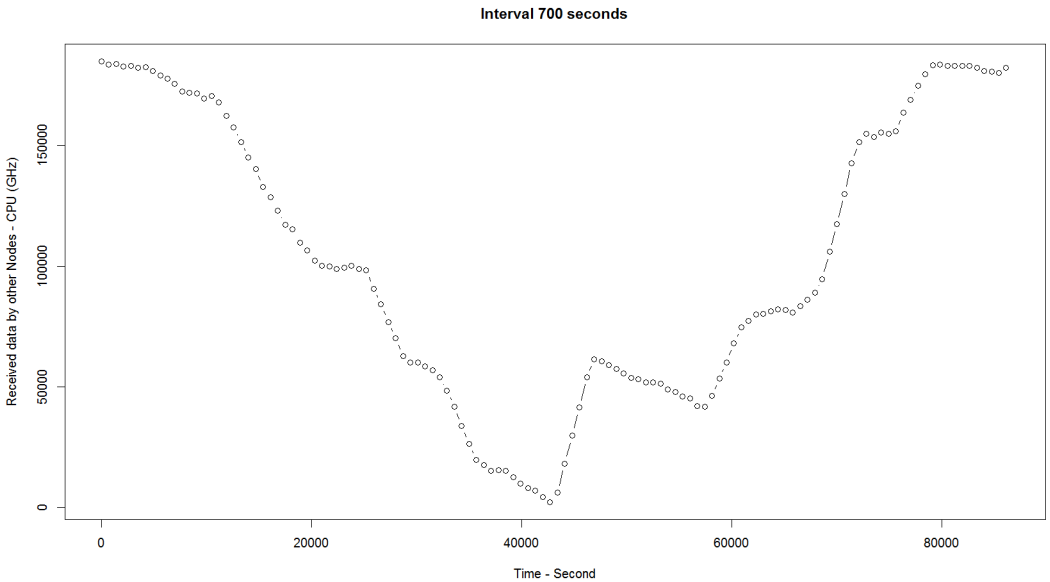

Figure D.7: How a cloud management system views cloud resources with a hold-down timer value 700 seconds, the updates points are plotted a circles.

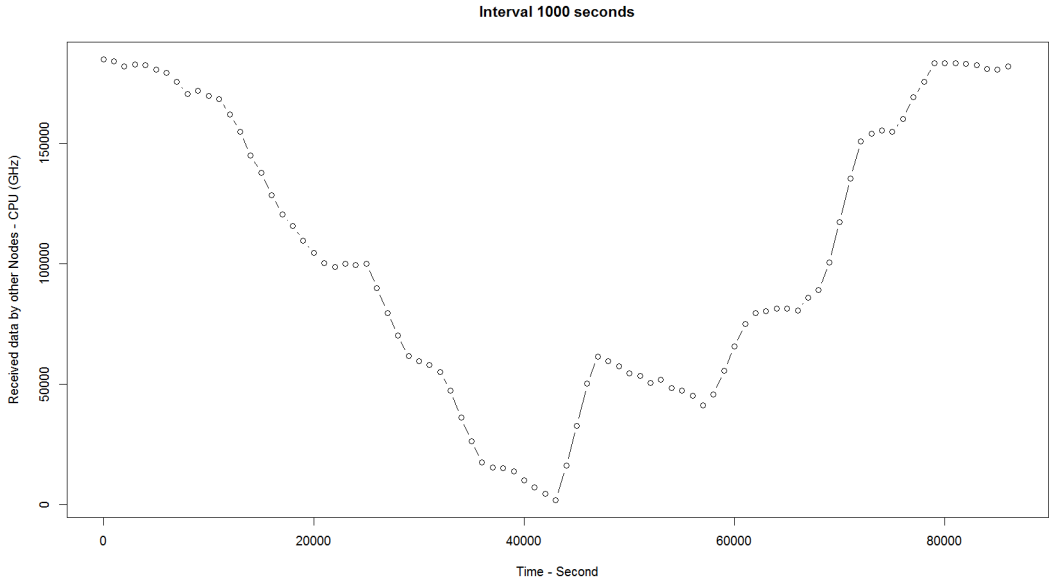

Figure D.8: How a cloud management system views cloud resources with a hold-down timer value 1000 seconds, the updates points are plotted a circles.

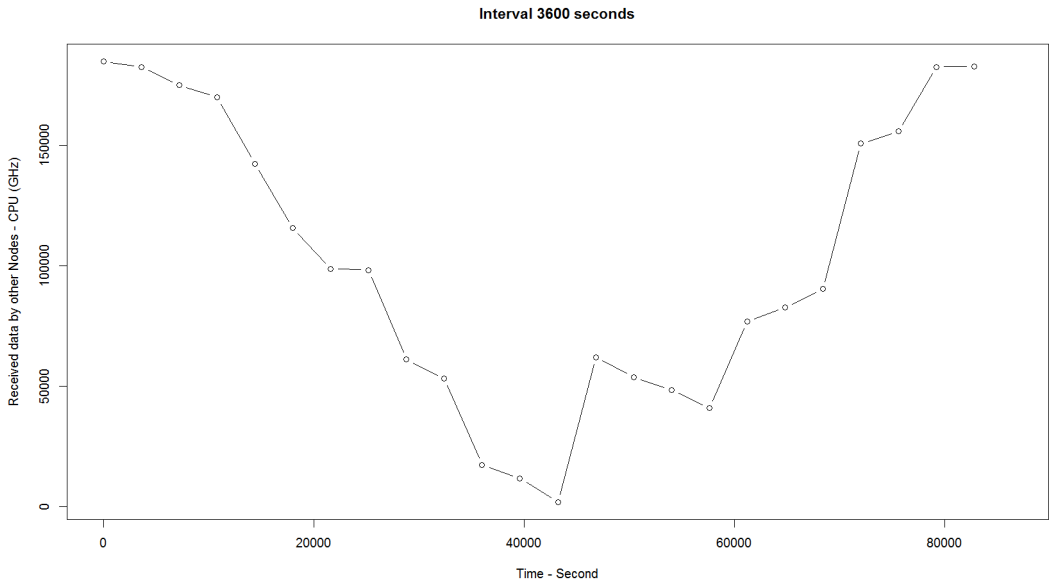

Figure D.9: How a cloud management system views cloud resources with a hold-down timer value 3600 seconds, the updates points are plotted a circles.

Appendix E

Equal-sized class-based update policy experiments

A key factor that effect on number of classes (and as a consequence number of updates) in an **equal-sized class-based update policy** is a base factor β (see section 2.5). This Appendix provides complete set of results when an equal-sized class-based update policy with different number of equal-sized classes (i.e., consequence of different β values) values was applied on a proposed simulated data center's resources (see section 5.3.1). It is worth mentioning that, the points that shown in circle in these graphs depict the time that the link-state update is sent out into the network. These Figures provide an overview to help understand how other node(s) in the network views the data center's resources based on each N_{eq} value.

Additionally, information about the number of updates per different number of equal-sized classes values based on simulated data center's resource changes (i.e., CPU, RAM, and storage) are given in this Appendix.

E.1 Data center's CPU

Table E.1: Equal-sized class-based update policy. Number of Cloud LSA updates based on changes in a proposed data center's **CPU**. In this Table "N_{eq}" refers to a *"number of equal-sized classes"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

N_{eq}	**Number of updates**					
	Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
2	9	12	7	3	61	11.449
4	17	19	14	5	65	13.139
6	22	24	19	7	71	13.424
9	36	40	32	9	127	22.088
12	44	48	39	13	131	23.390
15	50	55	46	17	133	23.435
20	65	70	60	25	171	25.531
30	84	89	78	38	174	27.790
50	117	124	111	51	211	32.273
80	194	201	186	123	293	38.453
120	318	327	309	245	489	46.383
200	623	635	611	493	798	60.505
400	1227	1244	1210	1039	1493	87.032
800	3004	3027	2980	2644	3215	118.043
1500	5943	5972	5914	5444	6213	145.513
2500	8743	8791	8694	8175	9154	245.546
5000	11649	11671	11628	11336	11837	109.197
8000	12734	12755	12714	12415	12971	103.831

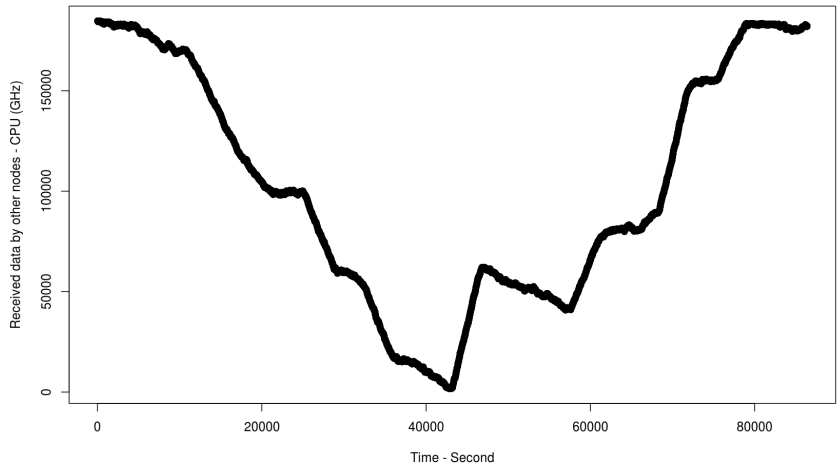

Figure E.1: Data center sample CPU (GHz) capacity per second

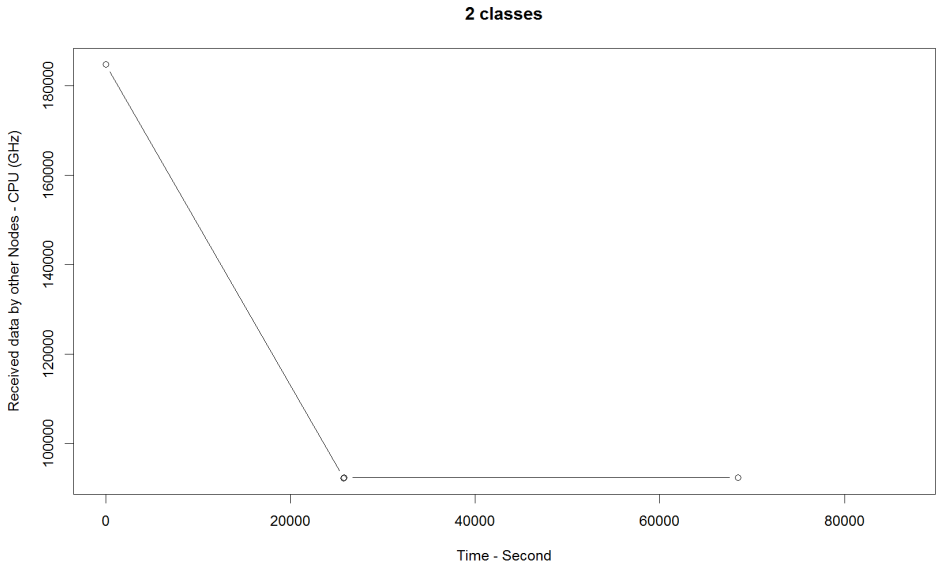

Figure E.2: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 2, the updates points are plotted a circles.

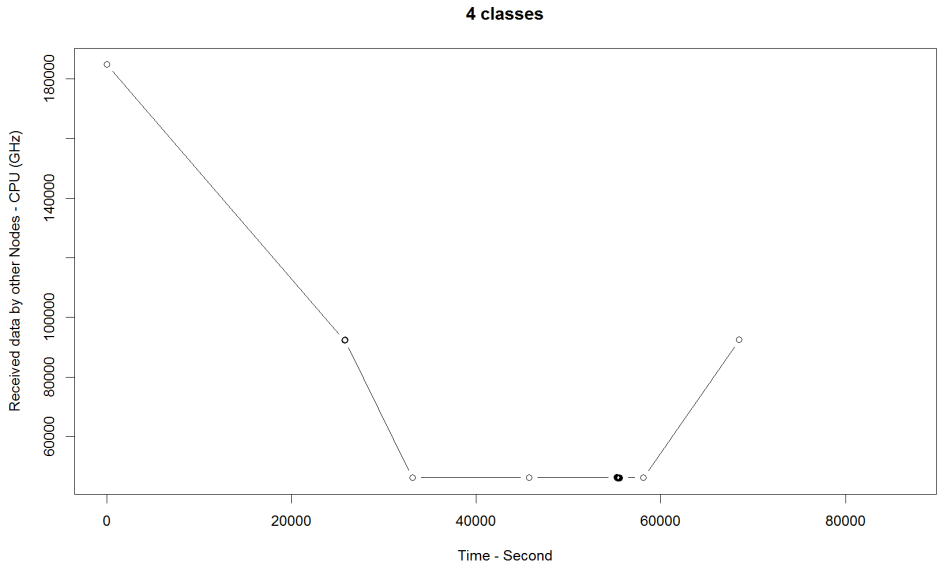

Figure E.3: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 4, the updates points are plotted a circles.

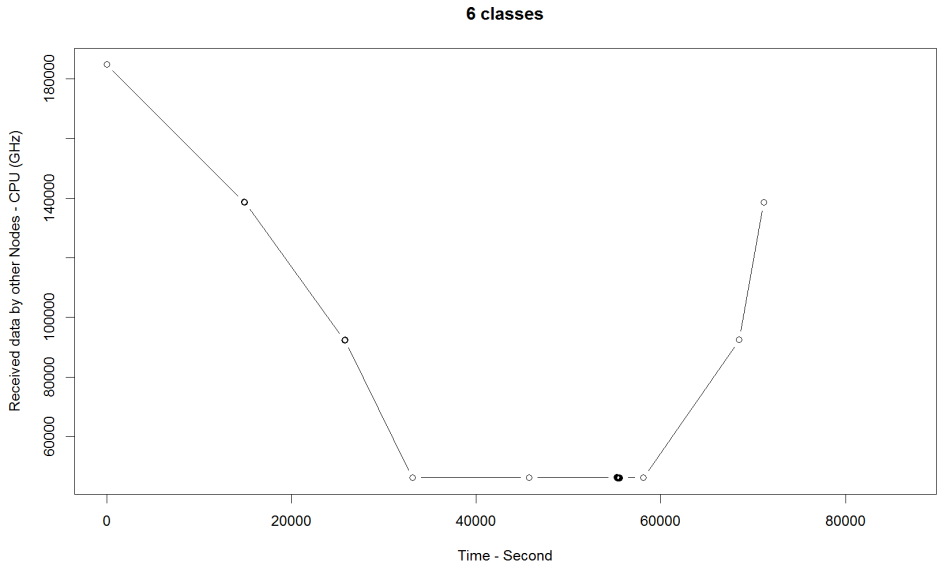

Figure E.4: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 6, the updates points are plotted a circles.

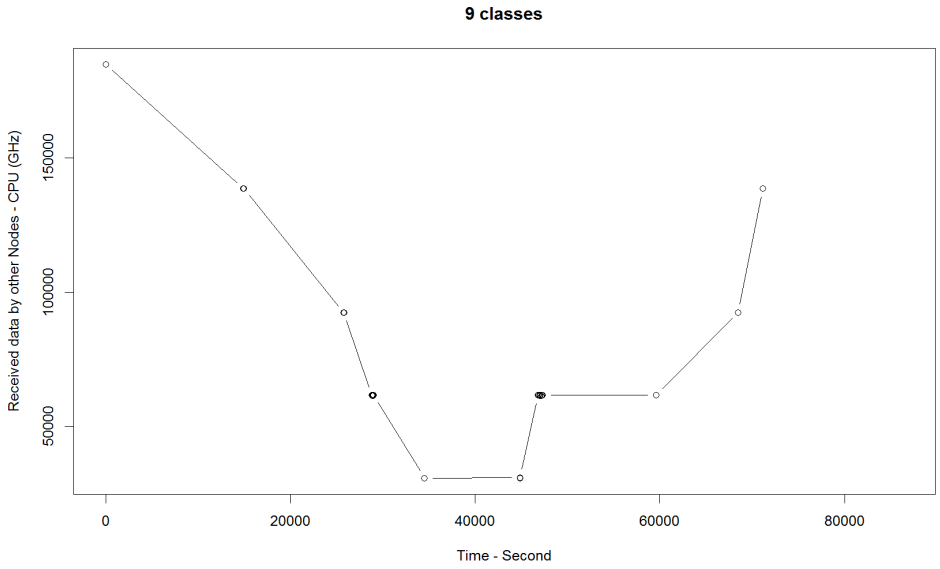

Figure E.5: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 9, the updates points are plotted a circles.

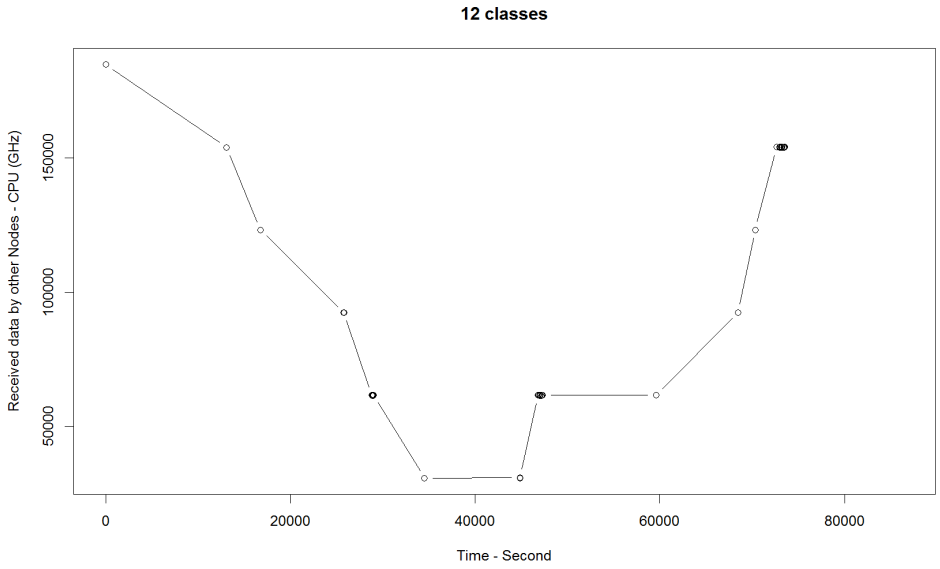

Figure E.6: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 12, the updates points are plotted a circles.

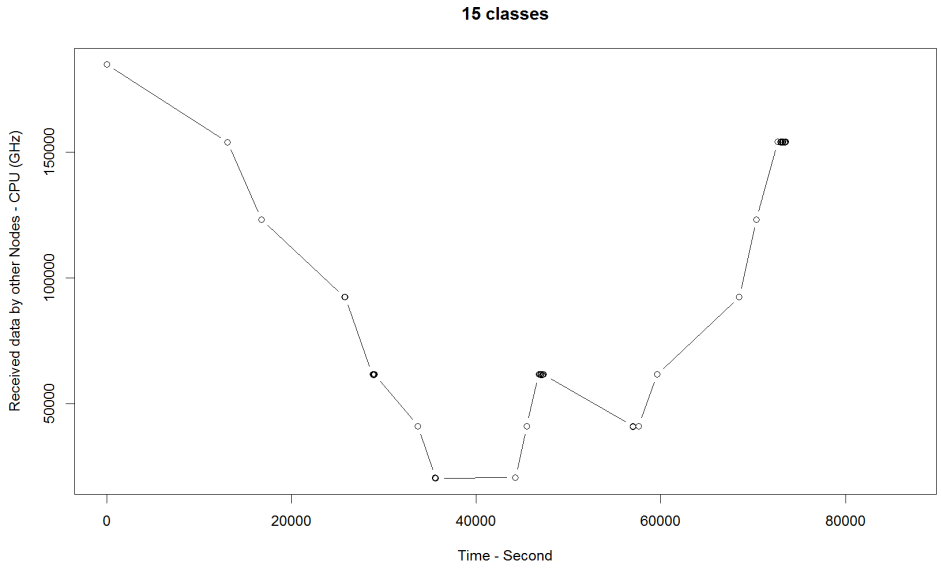

Figure E.7: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 15, the updates points are plotted a circles.

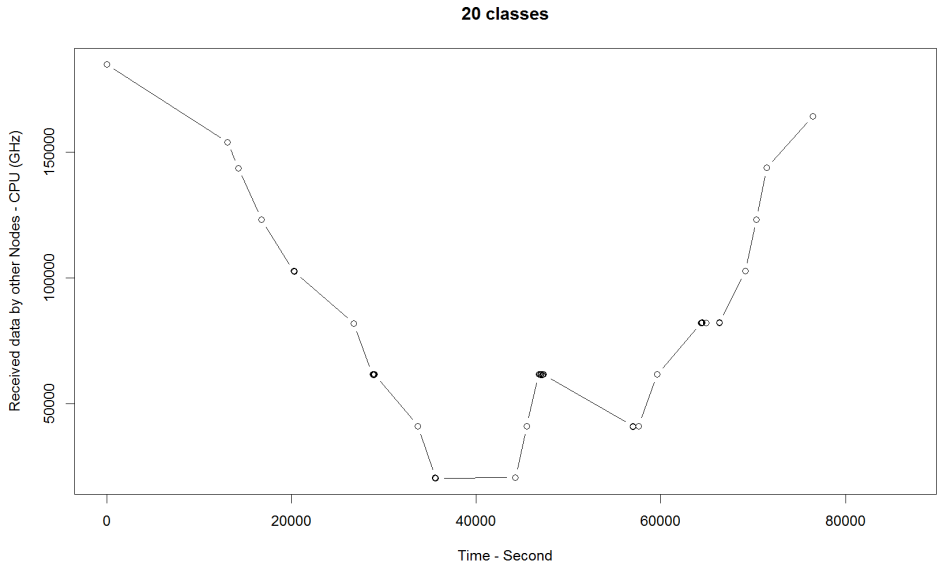

Figure E.8: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 20, the updates points are plotted a circles.

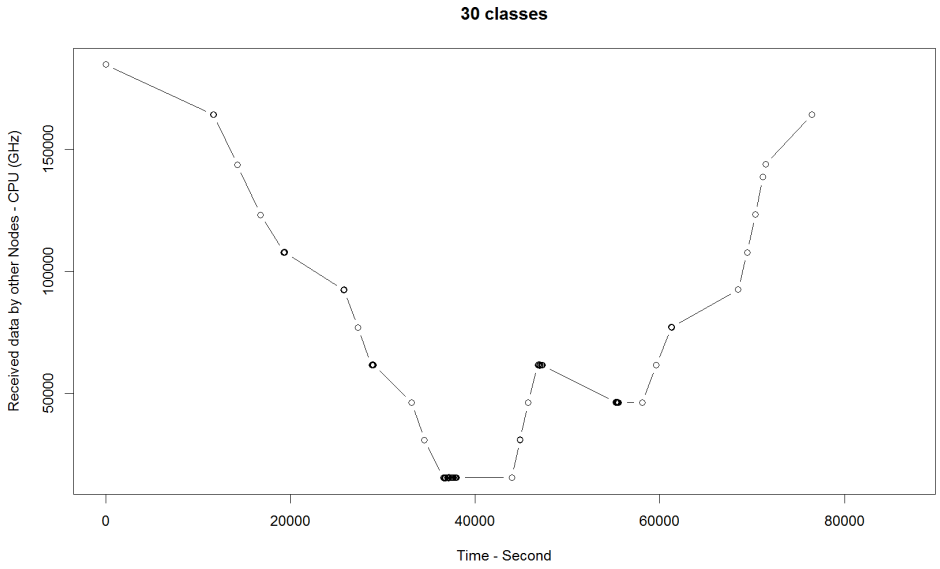

Figure E.9: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 30, the updates points are plotted a circles.

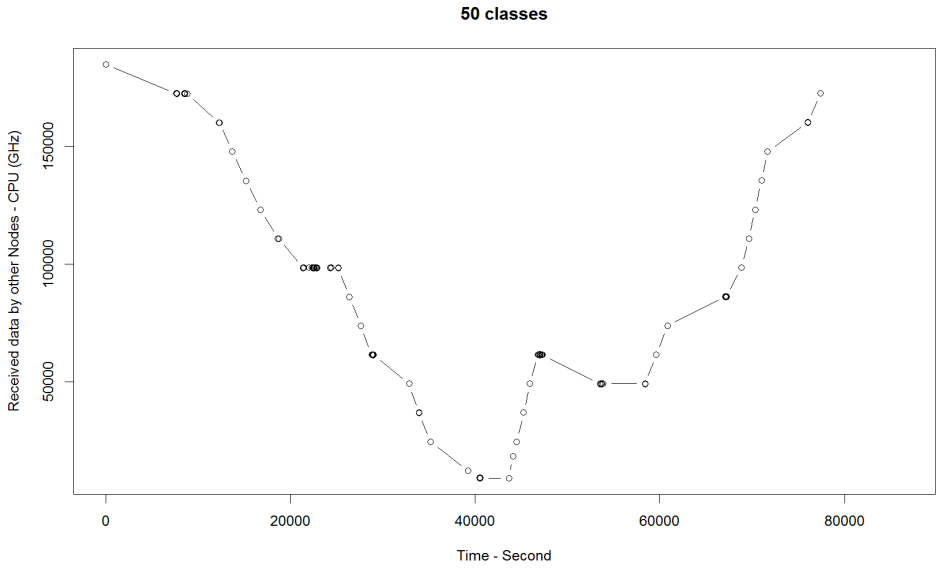

Figure E.10: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 50, the updates points are plotted a circles.

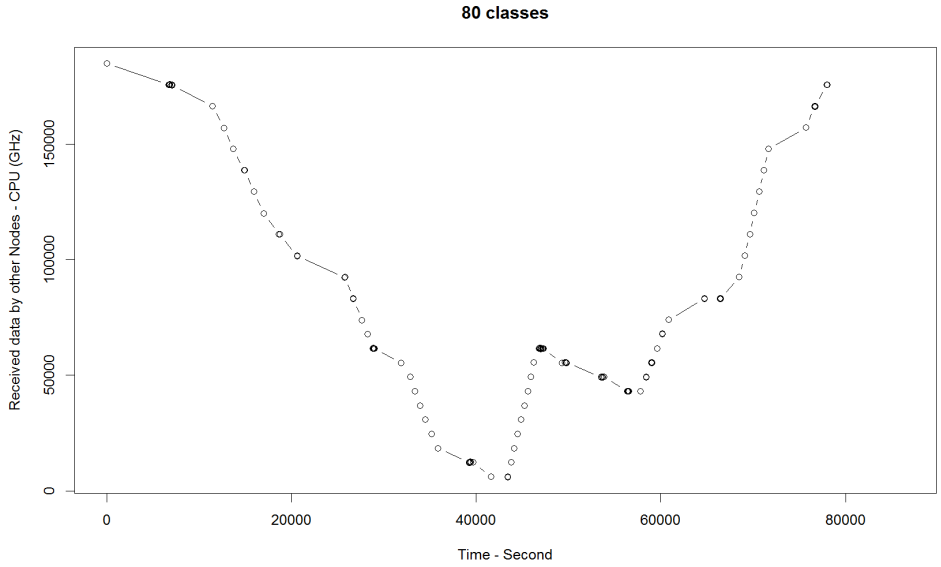

Figure E.11: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 80, the updates points are plotted a circles.

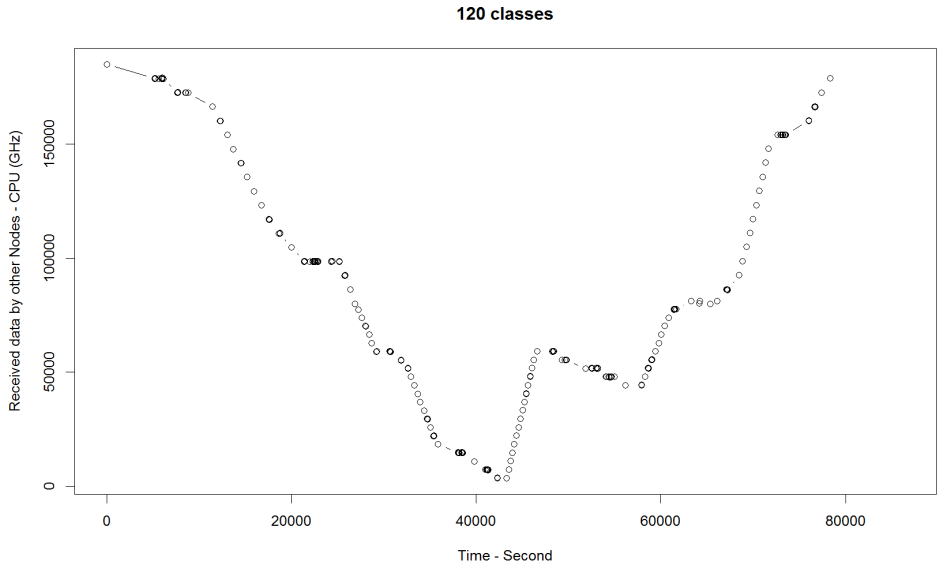

Figure E.12: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 120, the updates points are plotted a circles.

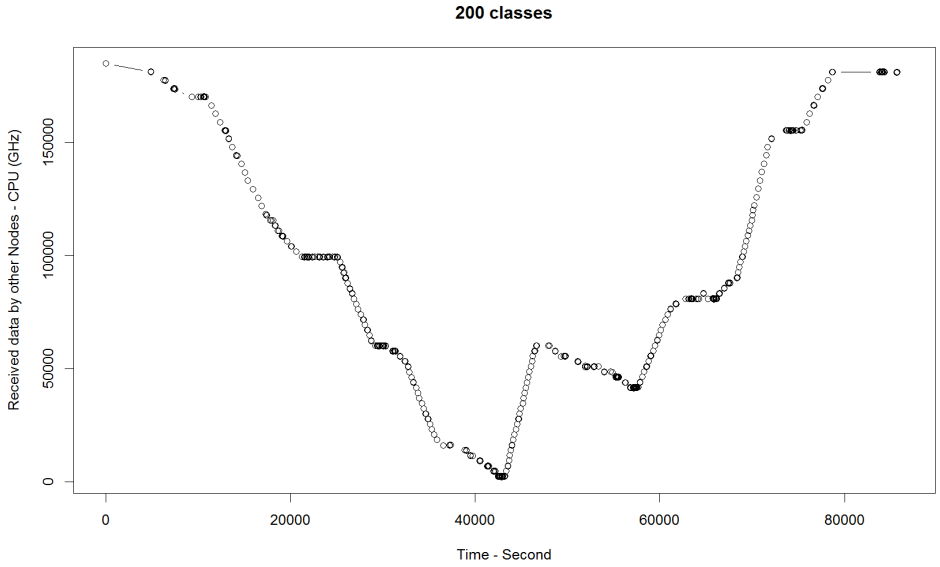

Figure E.13: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 200, the updates points are plotted a circles.

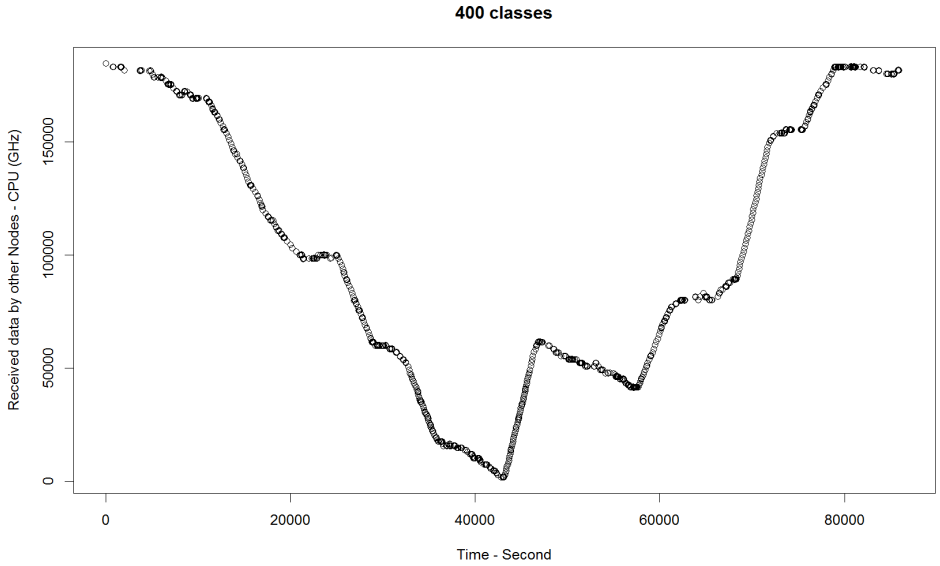

Figure E.14: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 400, the updates points are plotted a circles.

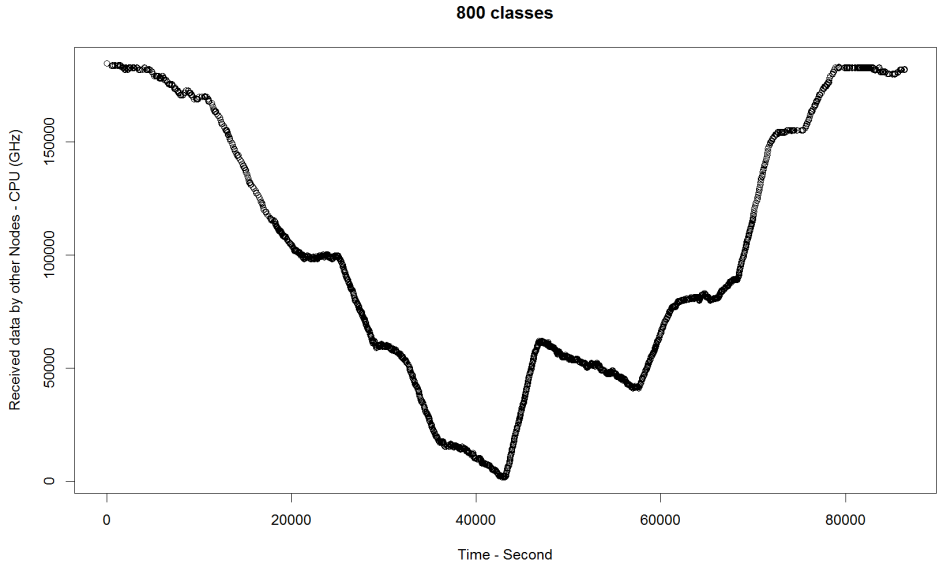

Figure E.15: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 800, the updates points are plotted a circles.

Figure E.16: Equal-sized class-based update policy. How a cloud management system views data center's *CPU* with a N_{eq} value 1500, the updates points are plotted a circles.

E.2 Data center's RAM

Table E.2: Equal-sized class-based update policy. Number of Cloud LSA updates based on changes in a proposed data center's **RAM**. In this Table "N_{eq}" refers to a *"number of equal size classes"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

N_{eq}	**Number of updates**					
	Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
2	11	14	8	3	109	14.630
4	17	20	14	5	113	16.280
6	22	26	19	7	117	17.738
9	39	45	34	9	177	27.600
12	46	51	40	13	179	27.064
15	51	57	46	13	149	27.346
20	62	67	57	23	139	24.573
30	84	90	78	32	210	30.205
50	113	120	107	53	237	32.480
80	193	202	184	121	323	43.372
120	319	330	308	199	525	55.533
200	611	623	598	483	759	61.415
400	1210	1226	1194	1029	1418	80.597
800	2961	2983	2939	2640	3169	111.179
1500	5928	5955	5901	5581	6223	135.596
2500	8790	8837	8744	8278	9237	233.969
5000	11769	11790	11749	11529	11955	102.065
8000	12835	12853	12817	12601	13062	90.791

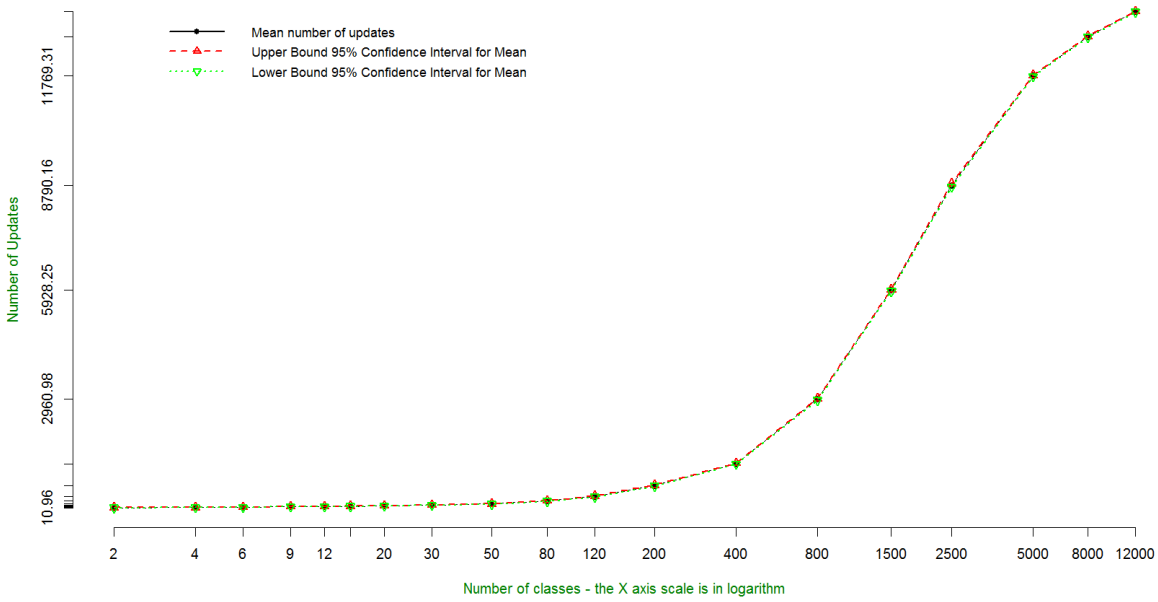

Figure E.17: Equal-sized class-based update policy. Number of updates per different number of classes N_{eq} values based on changes of a data center's *RAM* capacity. The "X" axis is in logarithm scale.

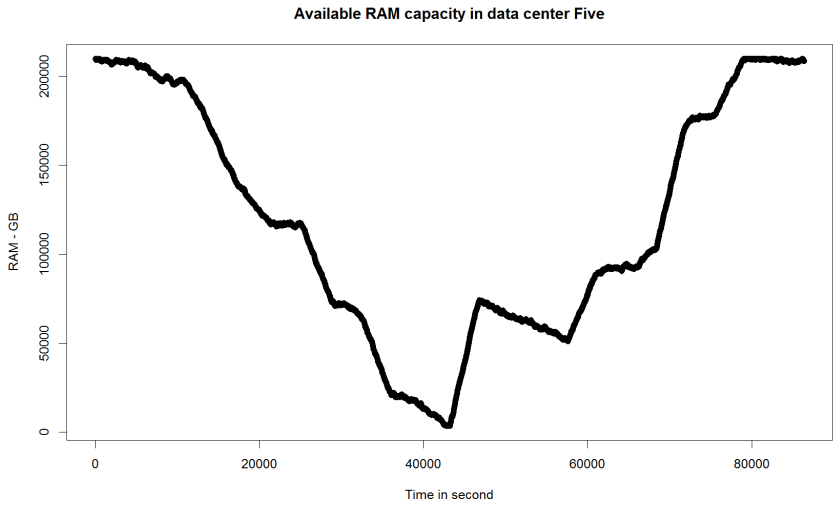

Figure E.18: Data center sample *RAM* (GB) capacity per second.

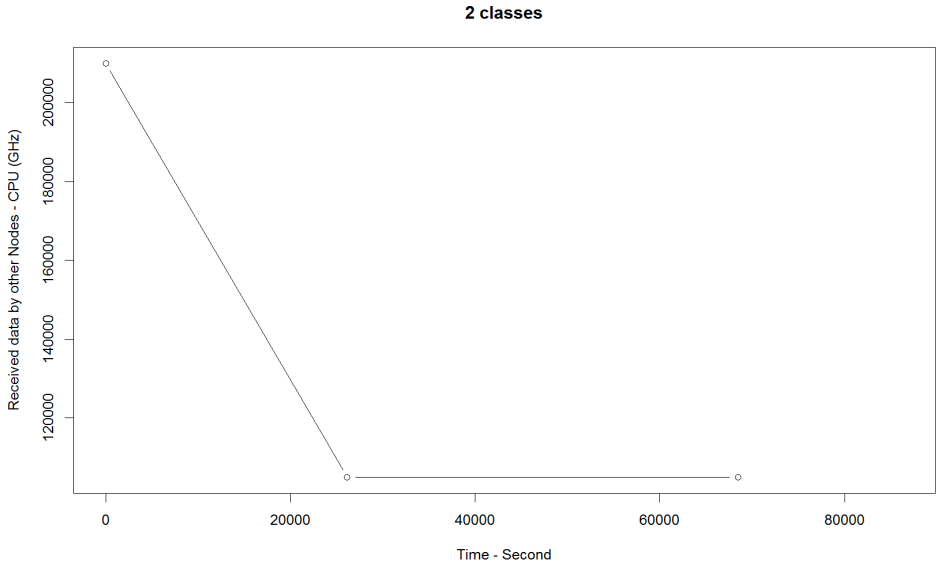

Figure E.19: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 2, the updates points are plotted a circles.

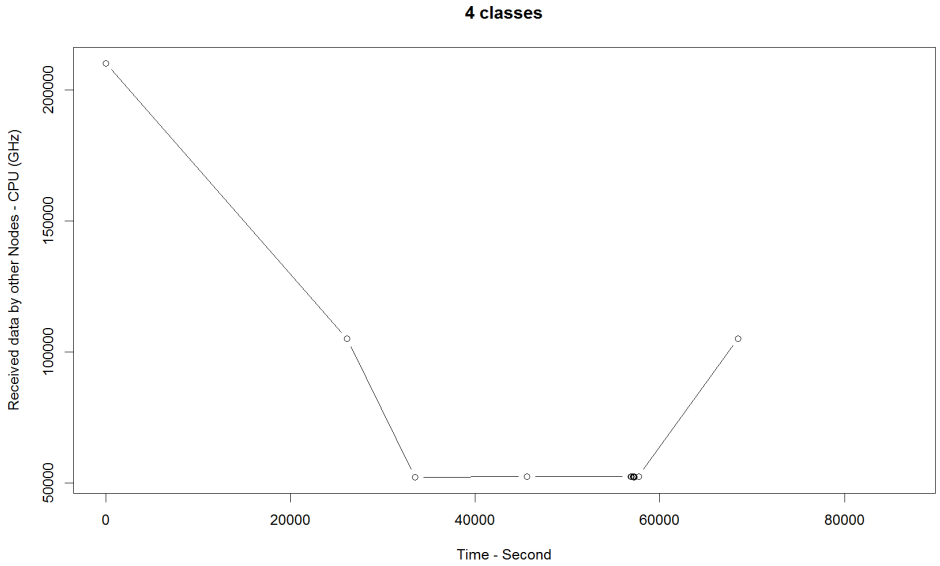

Figure E.20: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 4, the updates points are plotted a circles.

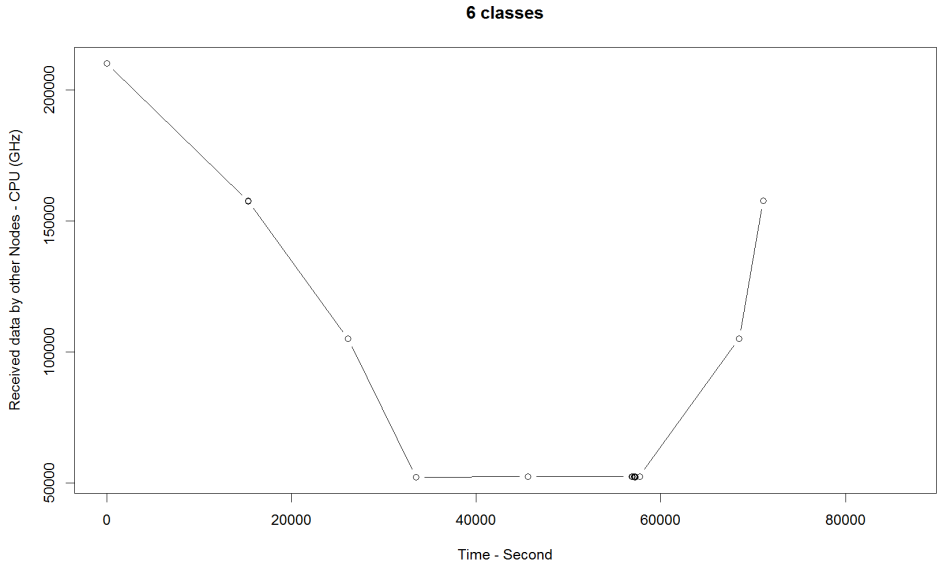

Figure E.21: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 6, the updates points are plotted a circles.

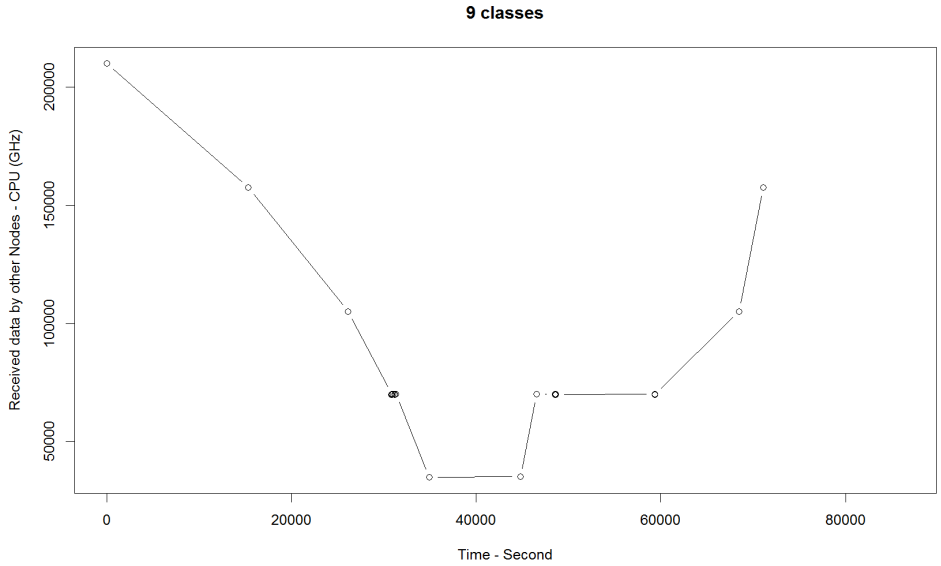

Figure E.22: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 9, the updates points are plotted a circles.

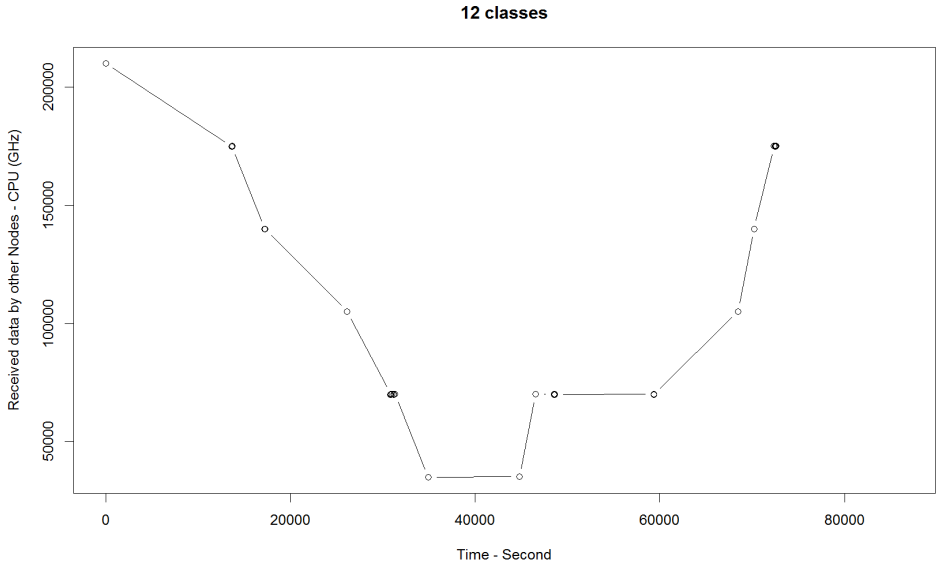

Figure E.23: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 12, the updates points are plotted a circles.

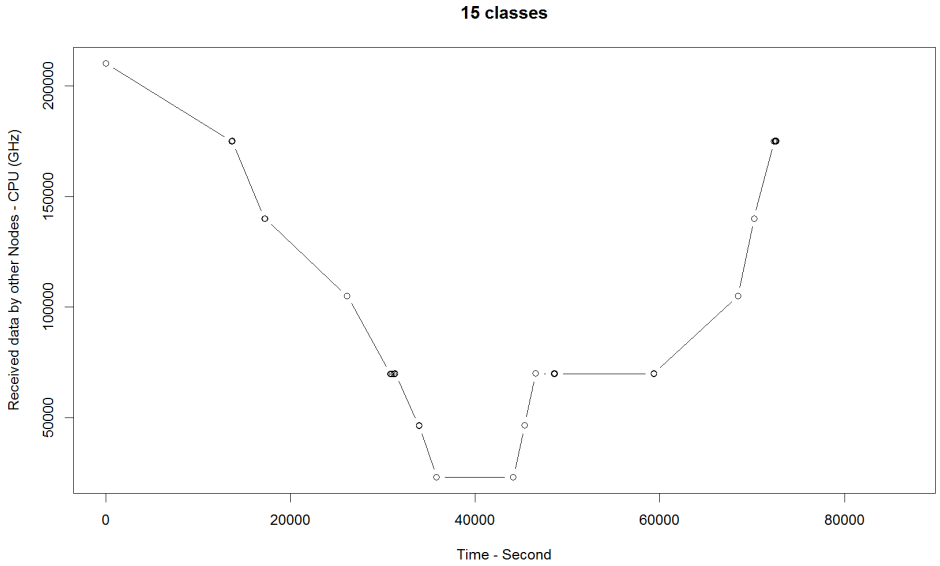

Figure E.24: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 15, the updates points are plotted a circles.

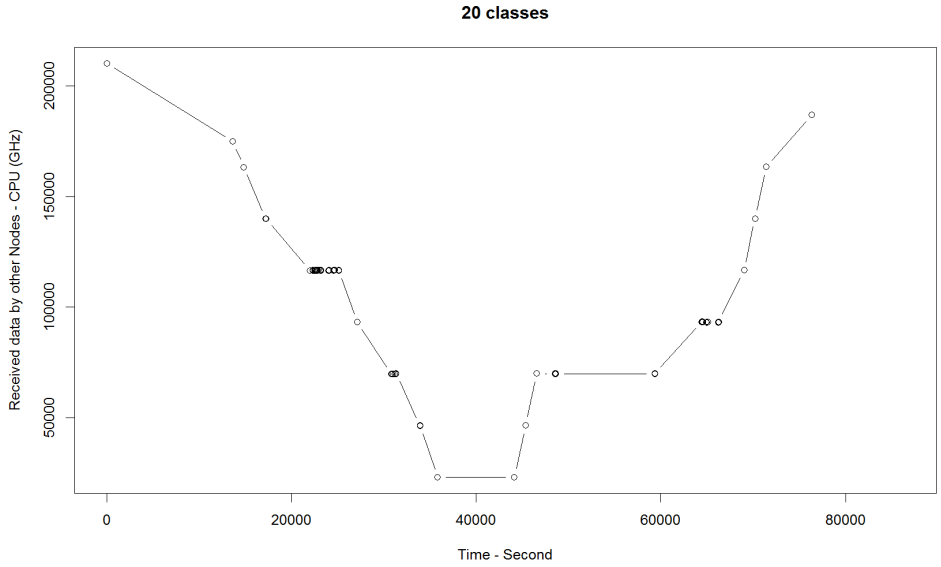

Figure E.25: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 20, the updates points are plotted a circles.

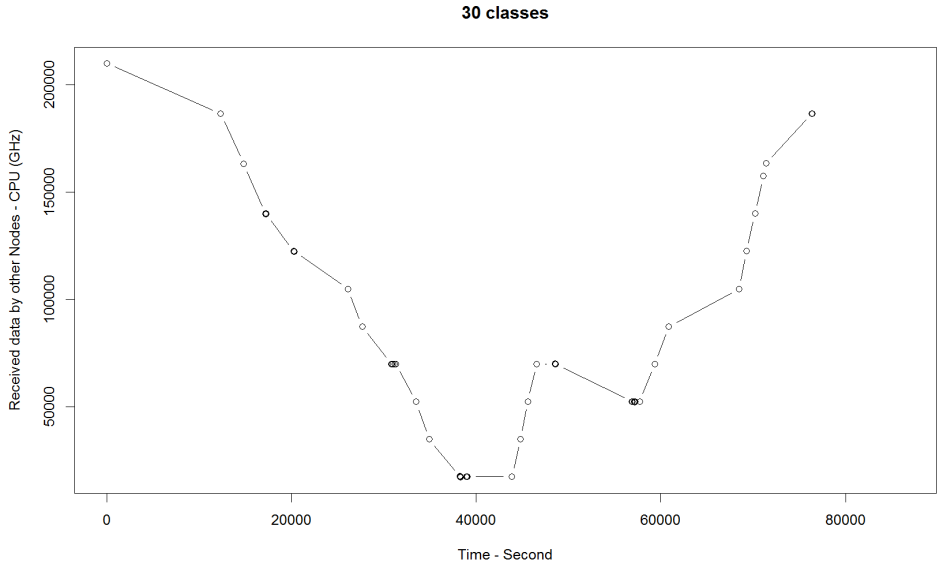

Figure E.26: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 30, the updates points are plotted a circles.

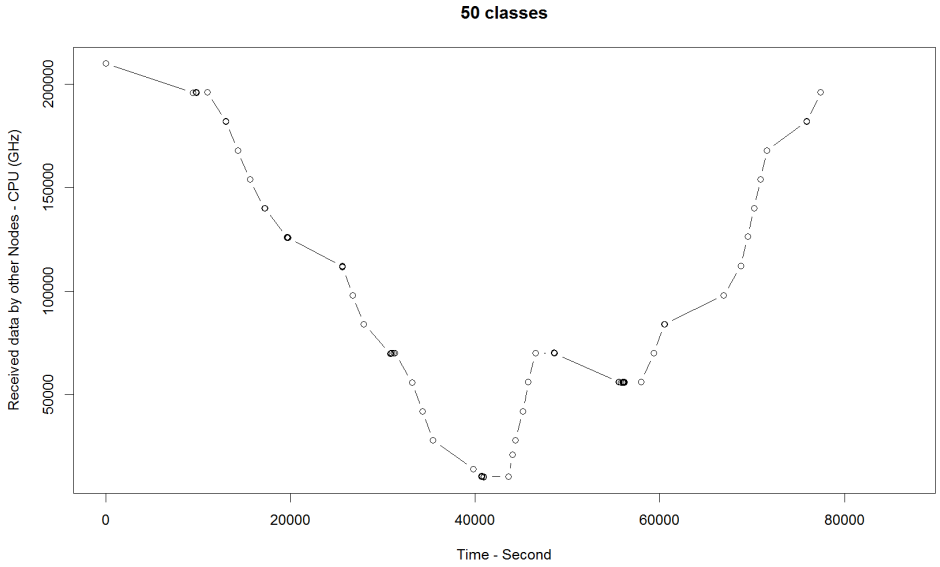

Figure E.27: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 50, the updates points are plotted a circles.

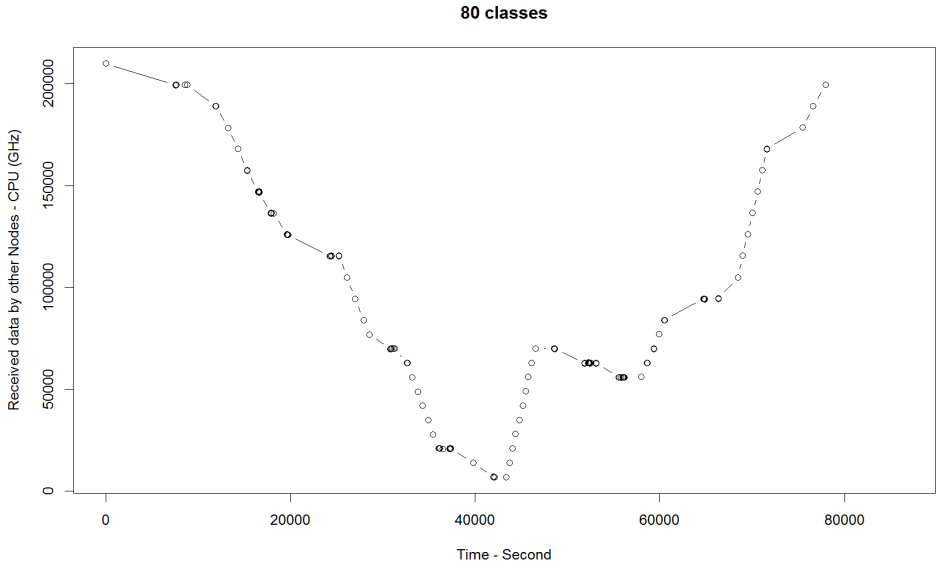

Figure E.28: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 80, the updates points are plotted a circles.

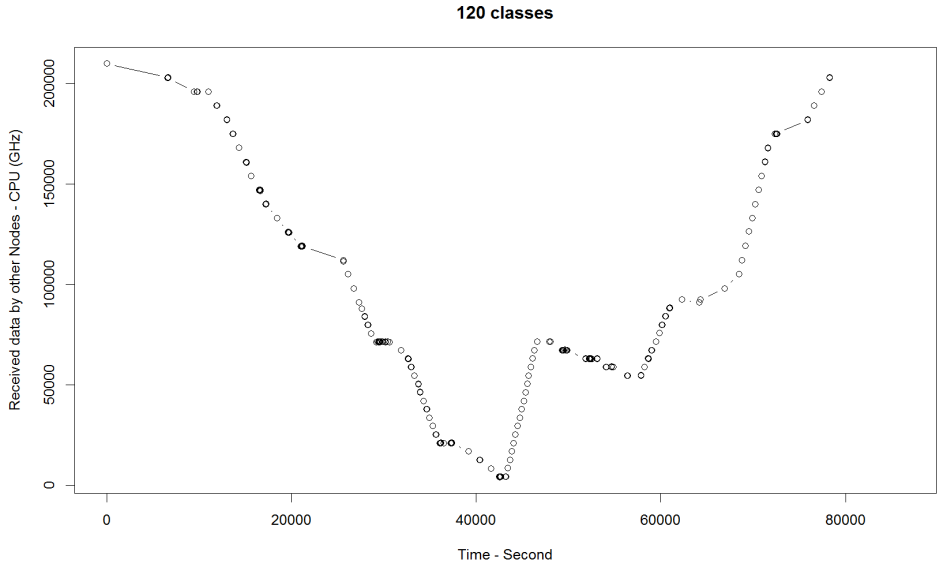

Figure E.29: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 120, the updates points are plotted a circles.

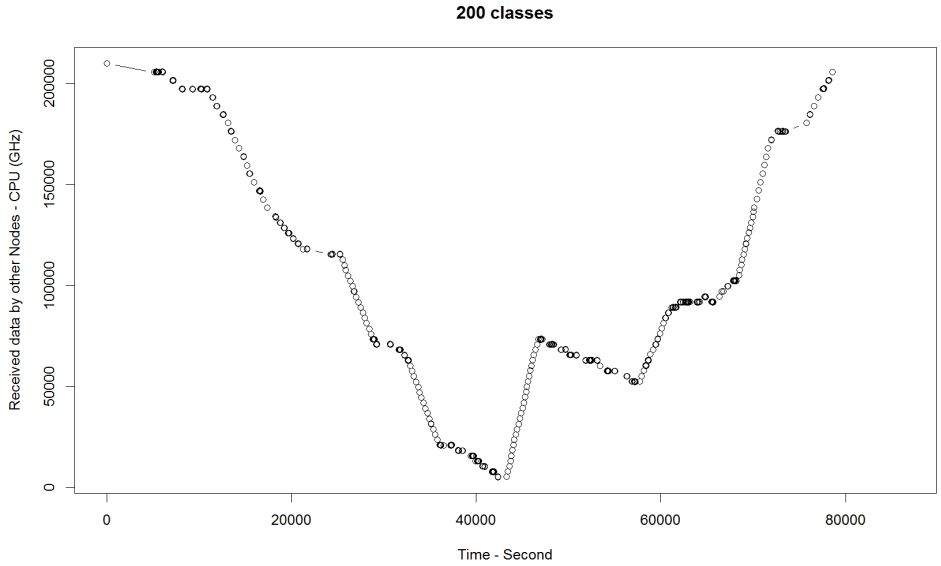

Figure E.30: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 200, the updates points are plotted a circles.

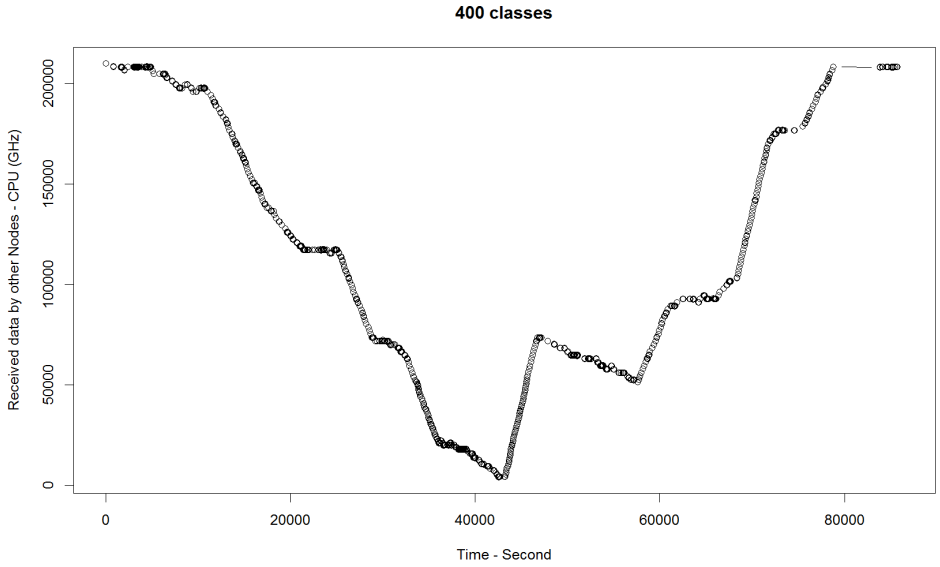

Figure E.31: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 400, the updates points are plotted a circles.

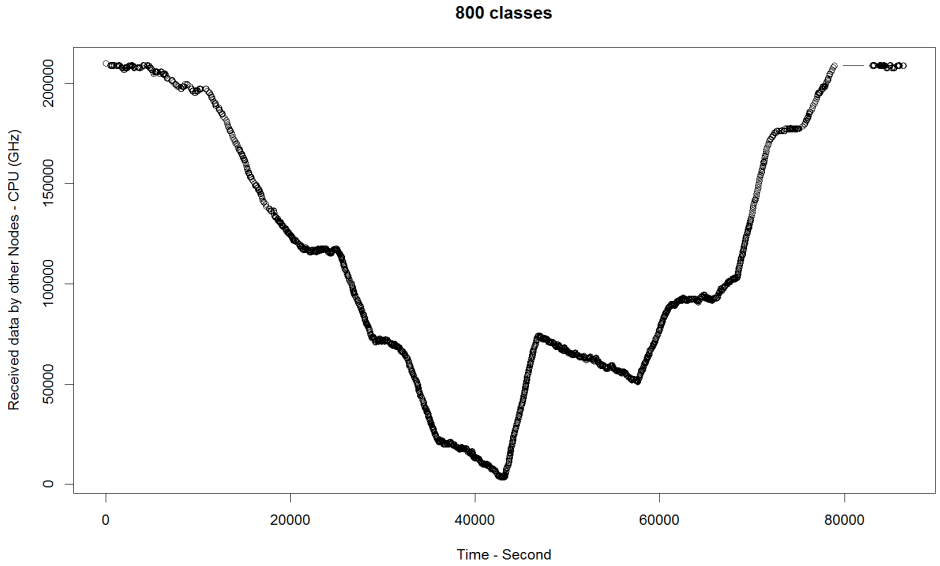

Figure E.32: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 800, the updates points are plotted a circles.

Figure E.33: Equal-sized class-based update policy. How a cloud management system views data center's *RAM* capacity with a N_{eq} value 1500, the updates points are plotted a circles.

E.3 Data center's storage

Table E.3: Equal-sized class-based update policy. Number of Cloud LSA updates based on changes in a proposed data center's **storage**. In this Table "N_{eq}" refers to a *"number of equal size classes"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

N_{eq}	**Number of updates**					
	Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
2	8	9	7	3	33	7.233
4	8	9	7	3	33	7.233
6	18	21	15	5	77	13.099
9	19	21	16	5	77	13.573
12	23	26	20	9	75	13.853
15	23	26	20	9	75	13.853
20	40	43	37	13	87	16.523
30	58	63	54	16	126	24.449
50	78	83	73	39	151	24.593
80	104	111	97	43	225	33.737
120	164	170	159	95	231	27.574
200	359	369	349	233	503	51.794
400	741	755	728	567	929	66.913
800	1771	1791	1750	1408	2030	101.231
1500	3767	3802	3733	3367	4098	174.767
2500	5660	5701	5619	5181	6229	204.999
8000	11481	11504	11458	11207	11696	115.604

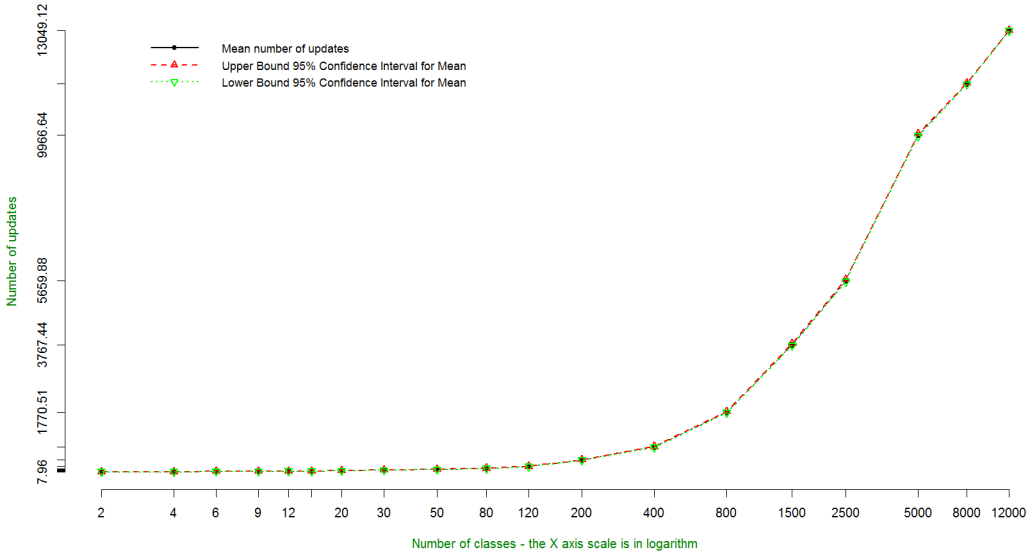

Figure E.34: Equal-sized class-based update policy. Number of updates per different number of classes N_{eq} values based on changes of a data center's *storage* capacity. The "X" axis is in logarithm scale.

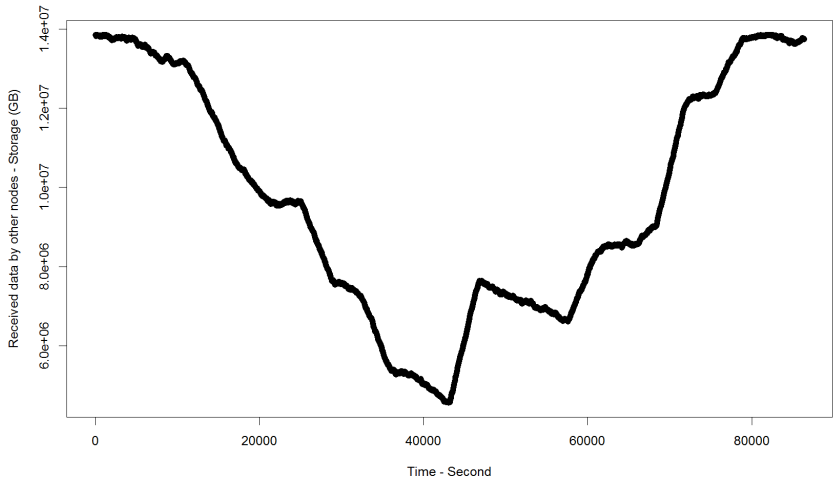

Figure E.35: Data center sample *storage* (GB) capacity per second.

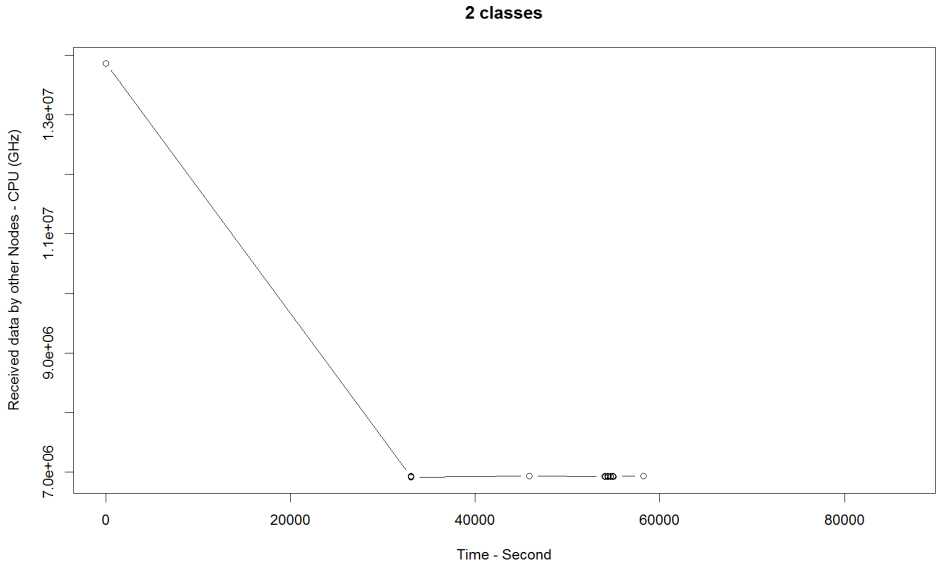

Figure E.36: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 2, the updates points are plotted a circles.

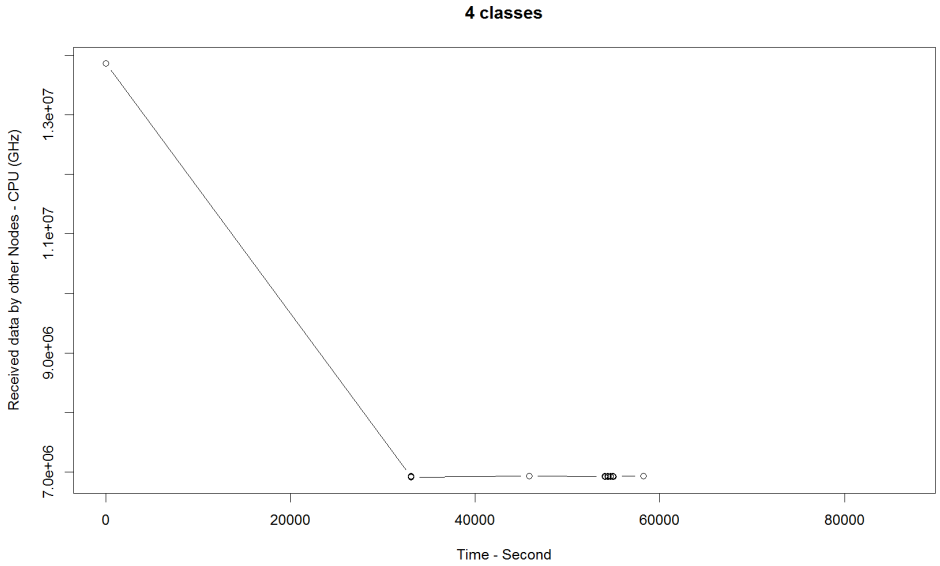

Figure E.37: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 4, the updates points are plotted a circles.

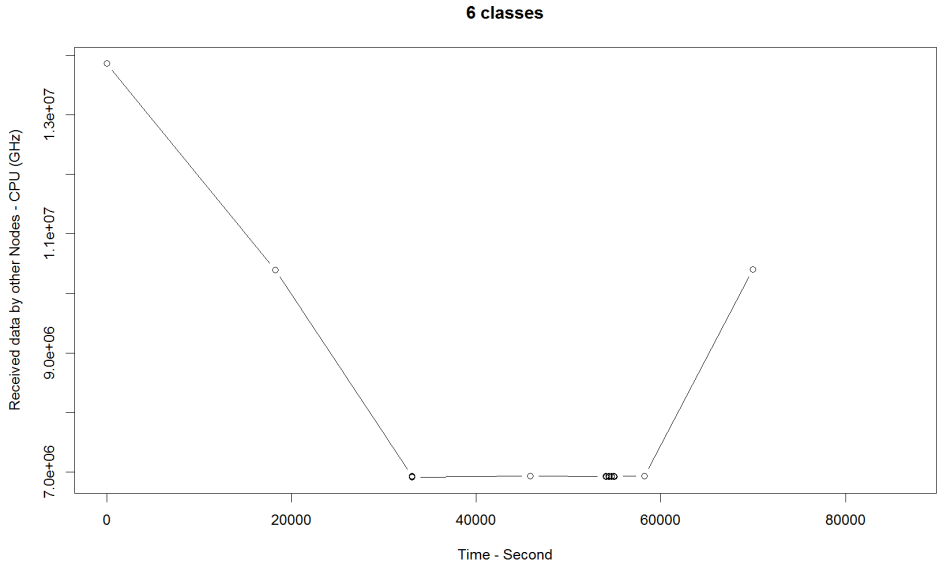

Figure E.38: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 6, the updates points are plotted a circles.

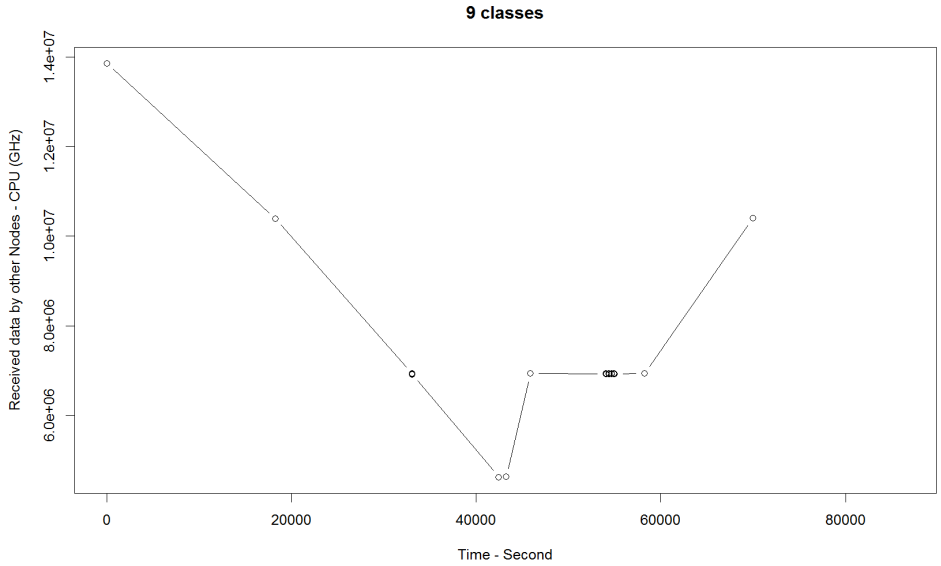

Figure E.39: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 9, the updates points are plotted a circles.

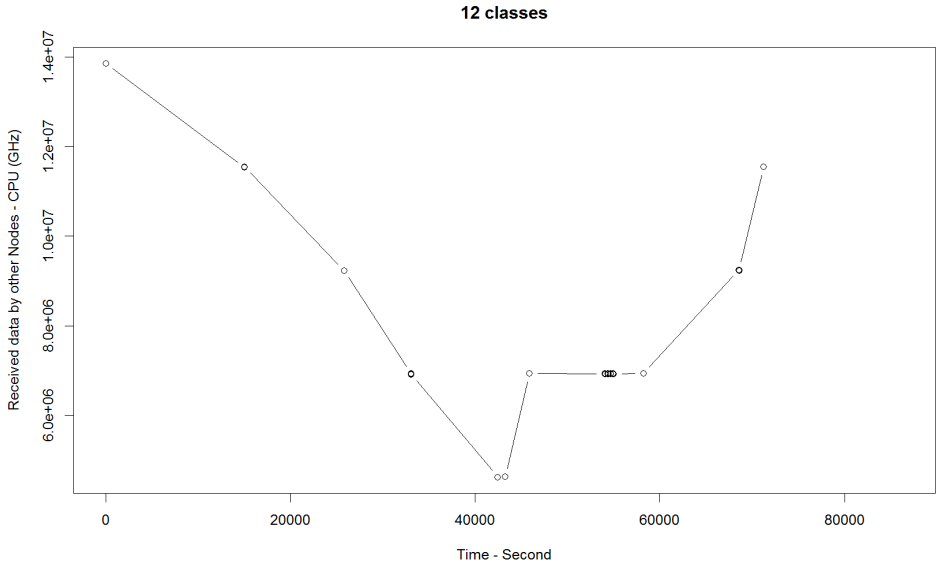

Figure E.40: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 12, the updates points are plotted a circles.

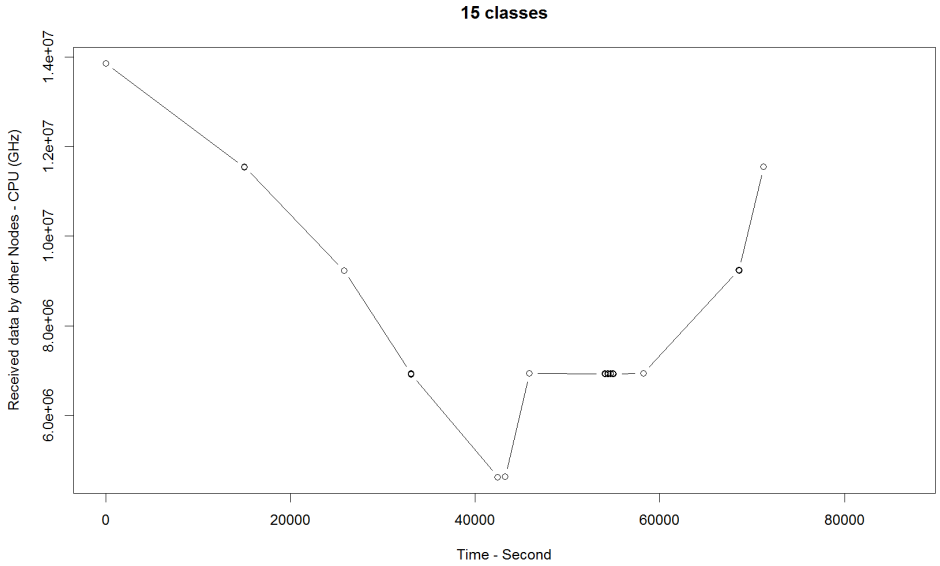

Figure E.41: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 15, the updates points are plotted a circles.

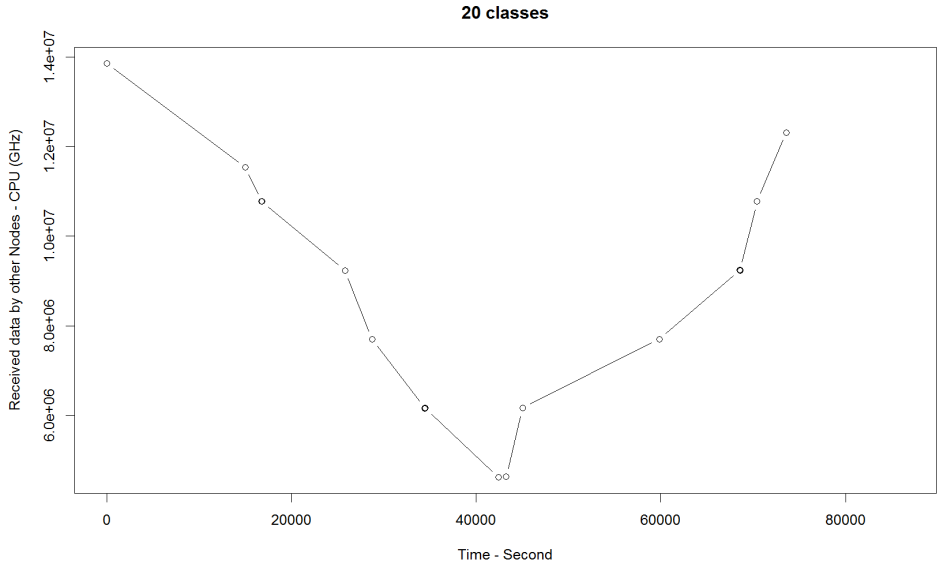

Figure E.42: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 20, the updates points are plotted a circles.

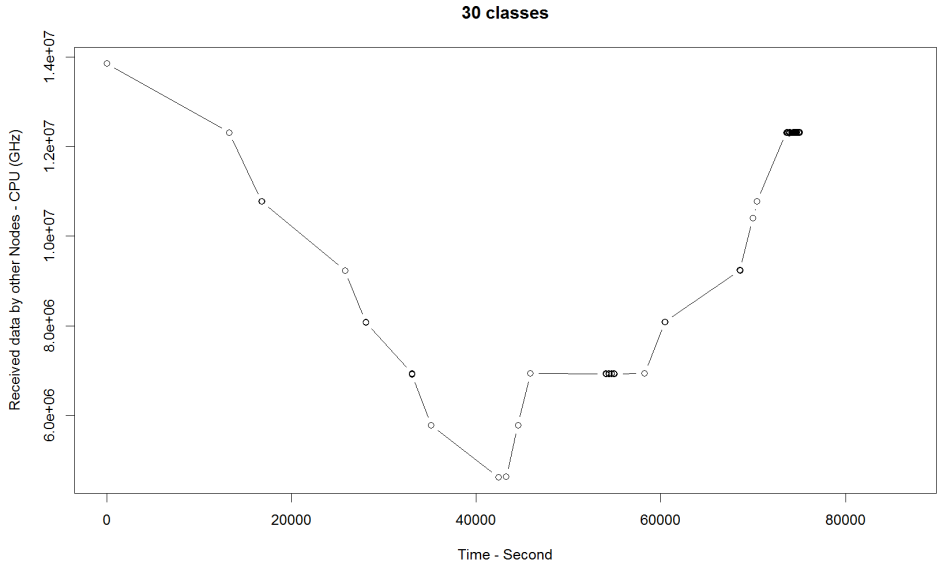

Figure E.43: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 30, the updates points are plotted a circles.

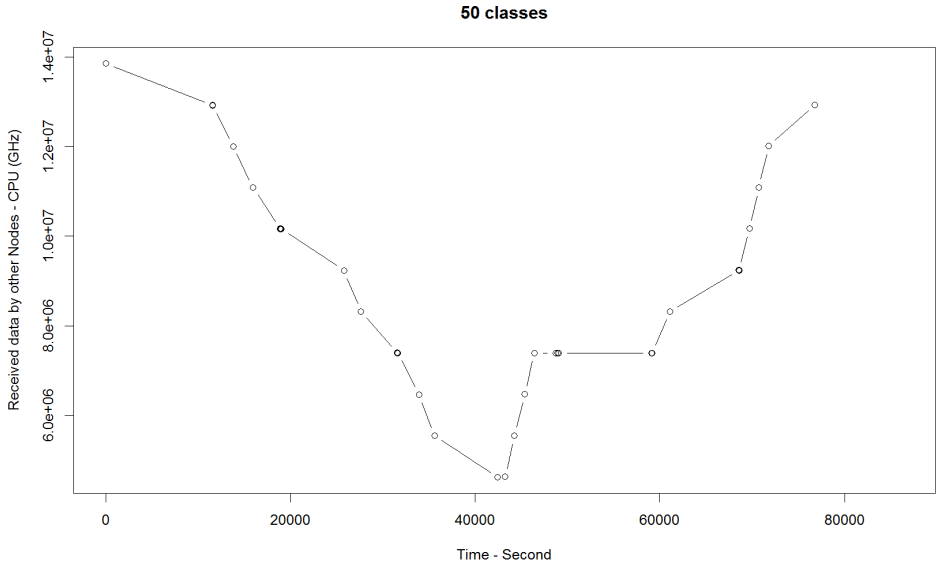

Figure E.44: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 50, the updates points are plotted a circles.

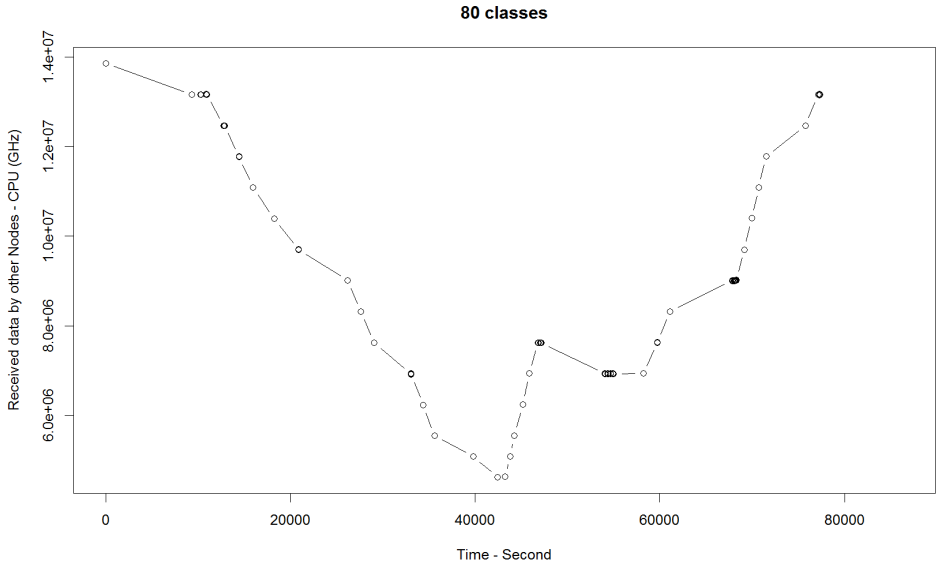

Figure E.45: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 80, the updates points are plotted a circles.

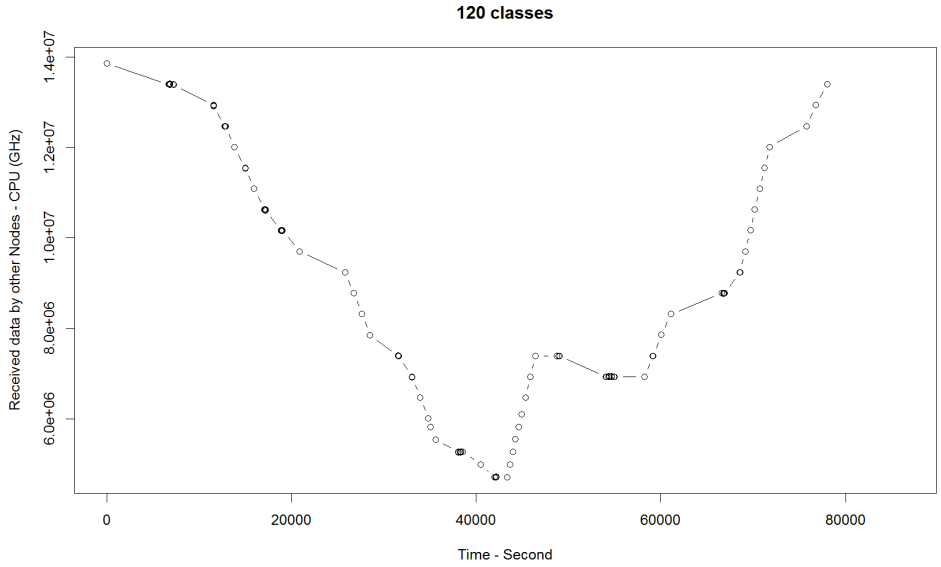

Figure E.46: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 120, the updates points are plotted a circles.

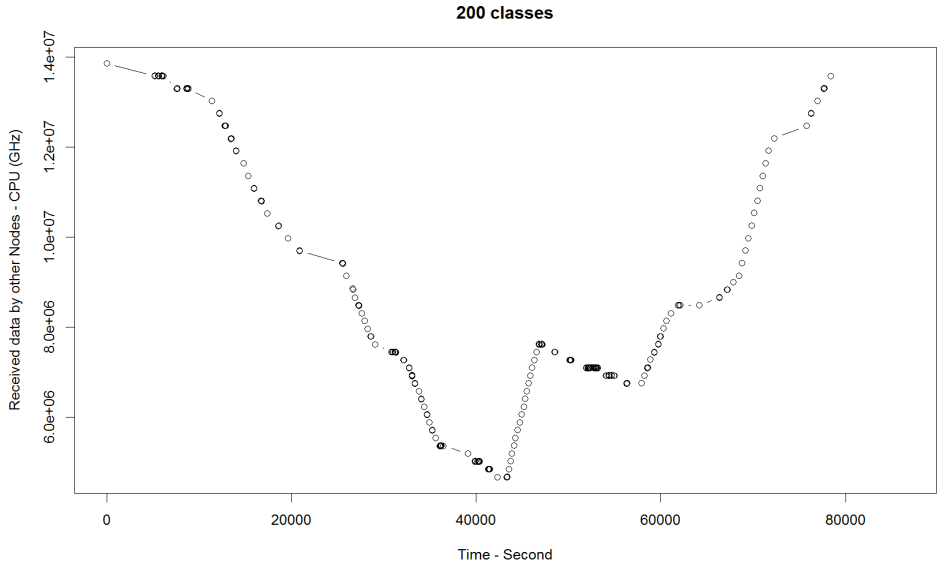

Figure E.47: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 200, the updates points are plotted a circles.

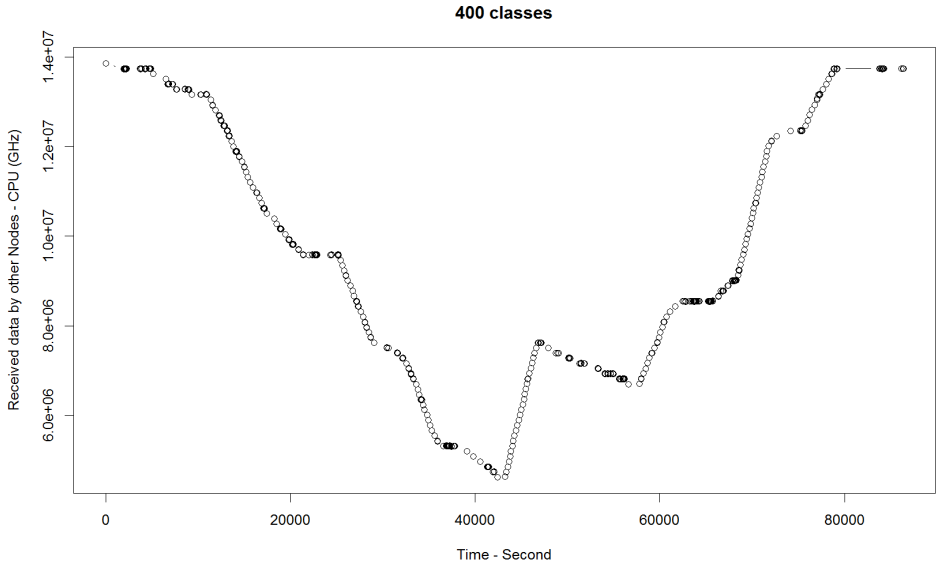

Figure E.48: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 400, the updates points are plotted a circles.

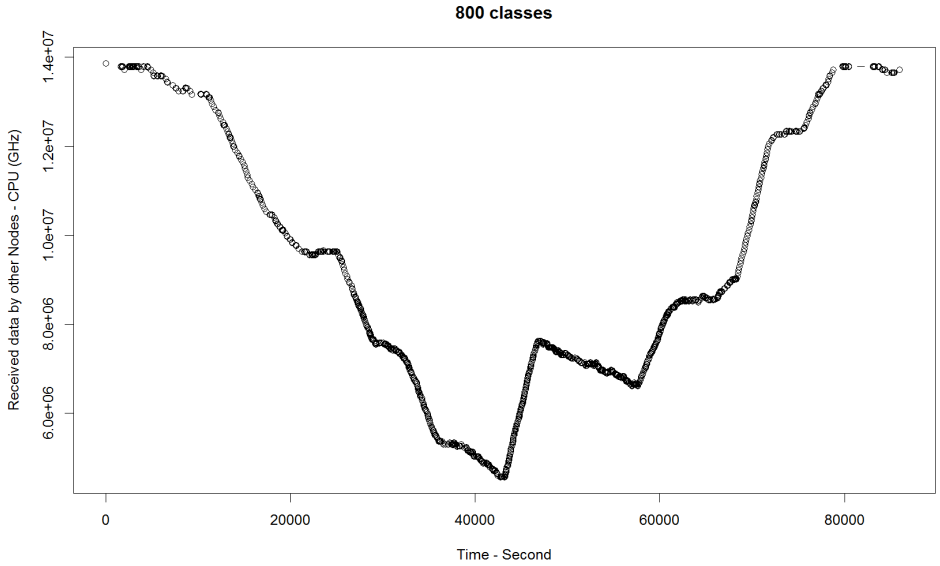

Figure E.49: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 800, the updates points are plotted a circles.

Figure E.50: Equal-sized class-based update policy. How a cloud management system views data center's *storage* capacity with a N_{eq} value 1500, the updates points are plotted a circles.

Appendix F

Exponential-sized class-based update policy

The two key factors that effect on number of classes (and as a consequence number of updates) in an exponential-sized class-based update policy are a base factor β and a growth factor f (see section 2.5. This Appendix provides complete set of results when an **exponential-sized class-based update policy** with different β and f values was applied on a proposed simulated data center's resources (see section 5.3.2). It is worth mentioning that, the points that shown in circle in these graphs depict the time that the link-state update is sent out into the network. These figures provide an overview to help understand how other node(s) in the network views the data center's resources based on each threshold value.

Additionally, information about the number of updates per different threshold value based on simulated data center's resource changes (i.e., CPU, RAM, and storage) are provided in this Appendix.

F.1 Data center's CPU

Table F.1: Exponential-sized class-based update policy. Number of Cloud LSA updates based on changes in proposed data center's **CPU**. In this Table "β" refers to a *"Base factor"*, "*f*" refers to *growth factor*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

β	*f*	**Number of updates**					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.00001	1.2	134	1	1	59	303	50.635
0.00001	1.4	80	1	1	27	206	36.339
0.00001	1.6	53	2	2	11	189	29.165
0.00001	1.8	41	2	2	11	179	27.503
0.00001	2	38	2	2	7	159	25.109
0.00005	1.2	132	1	1	53	295	47.644
0.00005	1.4	81	1	1	27	191	36.624
0.00005	1.6	50	2	2	13	111	22.871
0.00005	1.8	42	2	2	9	147	24.657
0.00005	2	42	2	2	9	148	25.844
0.0001	1.2	137	1	1	39	343	48.952
0.0001	1.4	79	1	1	21	229	36.992
0.0001	1.6	49	2	2	13	203	28.686
0.0001	1.8	44	2	2	11	205	31.216
0.0001	2	42	2	2	9	146	25.852
0.0005	1.2	129	1	1	55	323	50.596
0.0005	1.4	72	1	1	21	177	30.201

Continued on next page

Table F.1 – continued from previous page

β	f	Number of updates					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.0005	1.6	73	2	2	17	193	37.741
0.0005	1.8	43	2	2	13	149	23.694
0.0005	2	27	2	2	5	90	17.810
0.001	1.2	187	1	1	73	342	56.320
0.001	1.4	75	1	1	15	221	34.778
0.001	1.6	49	2	2	13	165	28.519
0.001	1.8	43	2	2	11	137	25.737
0.001	2	27	2	2	5	77	15.974
0.005	1.2	115	1	1	51	229	36.632
0.005	1.4	106	1	1	25	230	42.684
0.005	1.6	60	2	2	17	171	27.762
0.005	1.8	44	2	2	9	129	22.945
0.005	2	36	2	2	5	121	19.096
0.01	1.2	106	1	1	45	211	33.326
0.01	1.4	117	1	1	36	234	37.947
0.01	1.6	41	2	2	15	103	19.207
0.01	1.8	35	2	2	9	125	20.772
0.01	2	34	2	2	9	91	17.483
0.05	1.2	63	1	1	21	131	23.399
0.05	1.4	45	1	1	11	167	23.827
0.05	1.6	34	2	2	9	107	19.968
0.05	1.8	35	2	2	5	81	19.087
0.05	2	32	2	2	5	83	17.282

Continued on next page

Table F.1 – continued from previous page

β	f	Number of updates					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.1	1.2	113	1	1	28	270	52.684
0.1	1.4	29	1	1	10	87	17.602
0.1	1.6	29	2	2	9	120	17.950
0.1	1.8	31	2	2	7	116	19.122
0.1	2	23	2	2	5	71	15.536

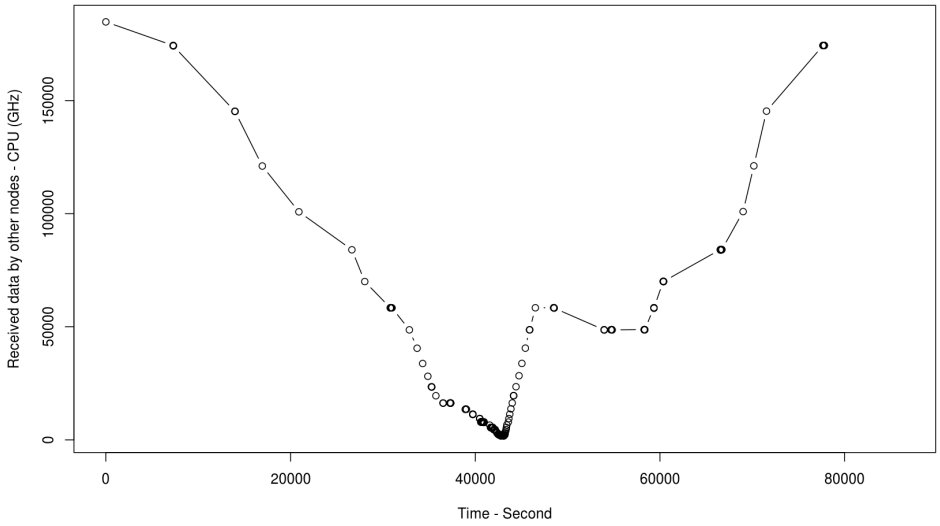

Figure F.1: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-05 and a growth factor f = 1.2.

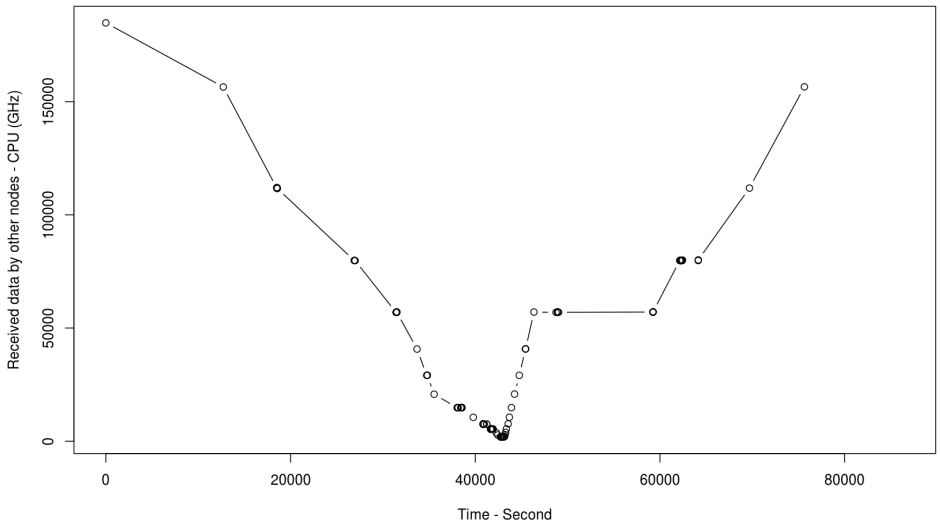

Figure F.2: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-05 and a growth factor f = 1.4.

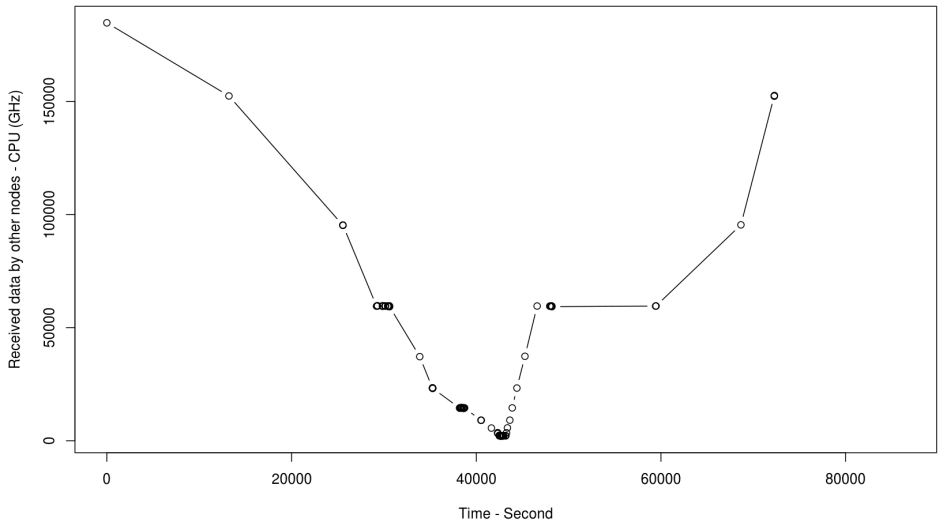

Figure F.3: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-05 and a growth factor f = 1.6.

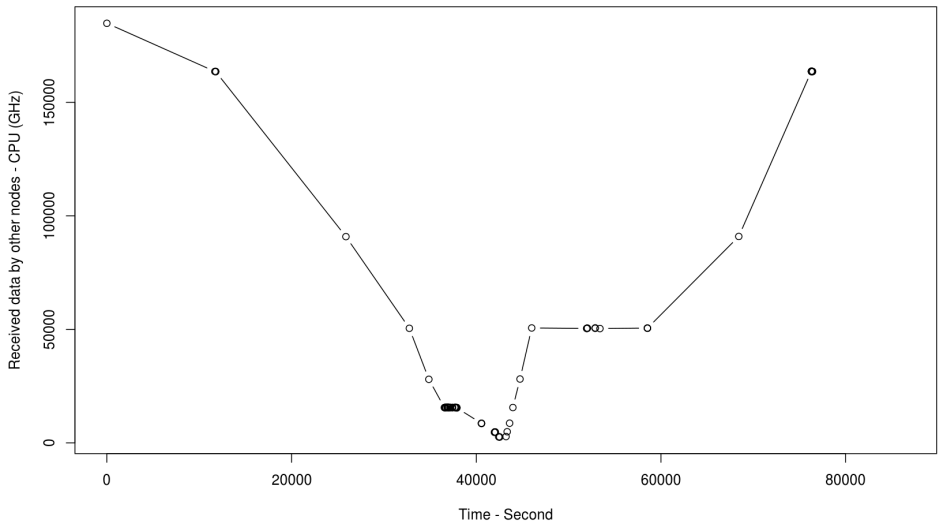

Figure F.4: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-05 and a growth factor f = 1.8.

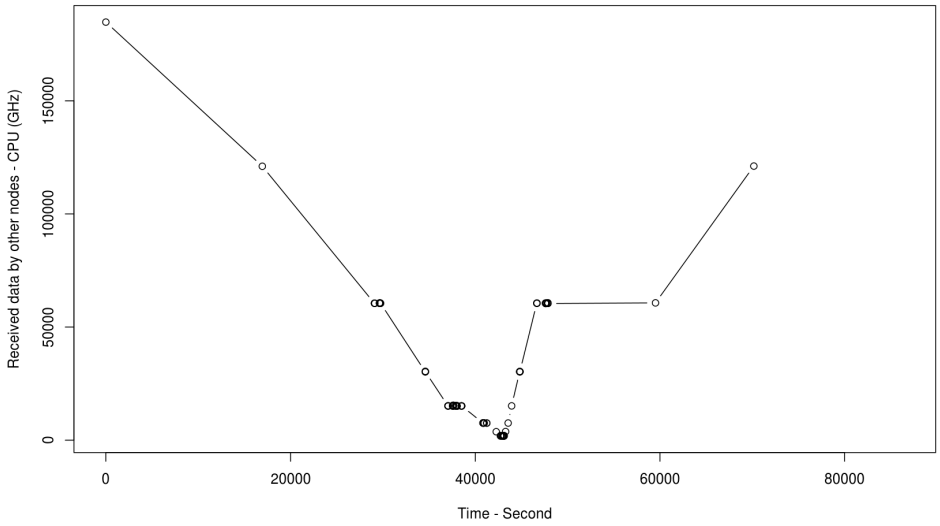

Figure F.5: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-05 and a growth factor f = 2.

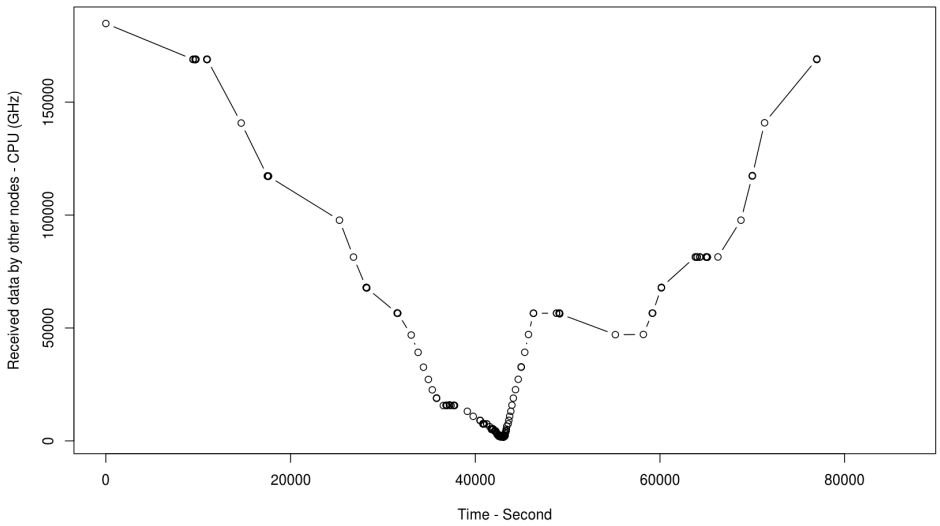

Figure F.6: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-05 and a growth factor f = 1.2.

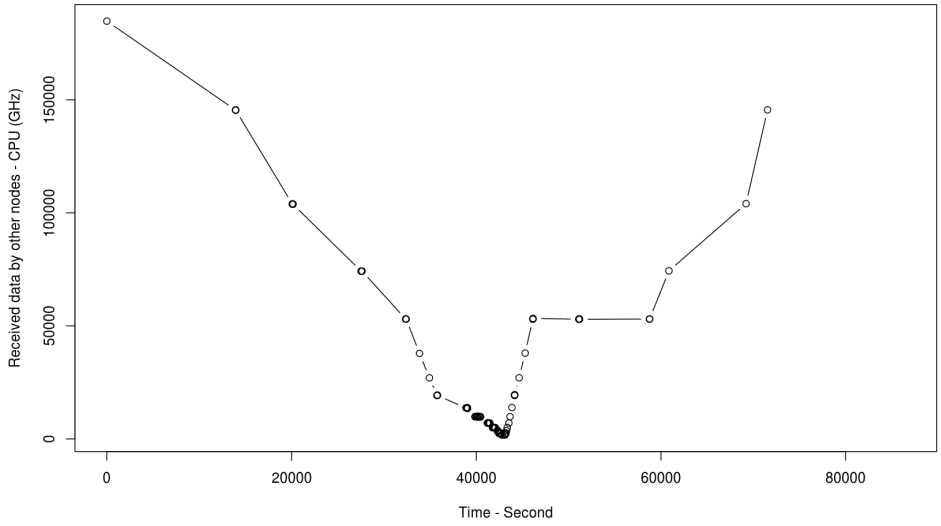

Figure F.7: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-05 and a growth factor f = 1.4.

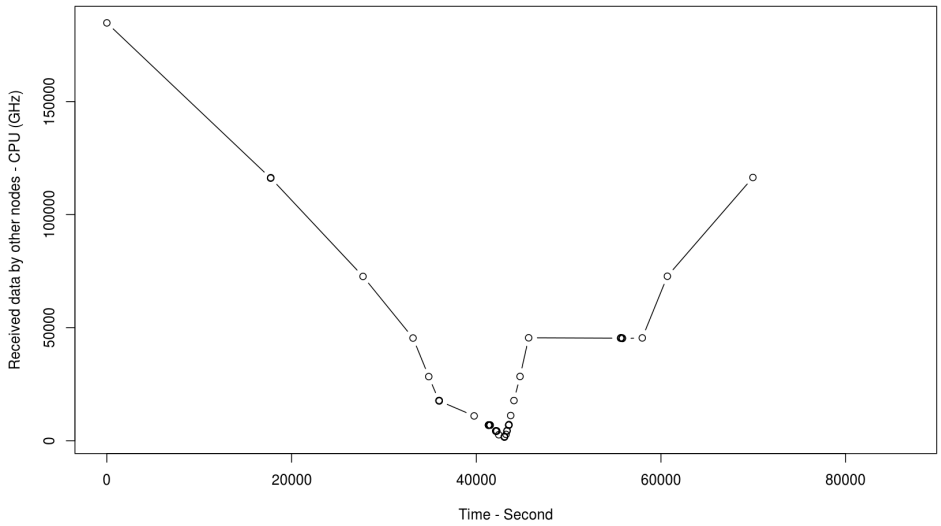

Figure F.8: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-05 and a growth factor f = 1.6.

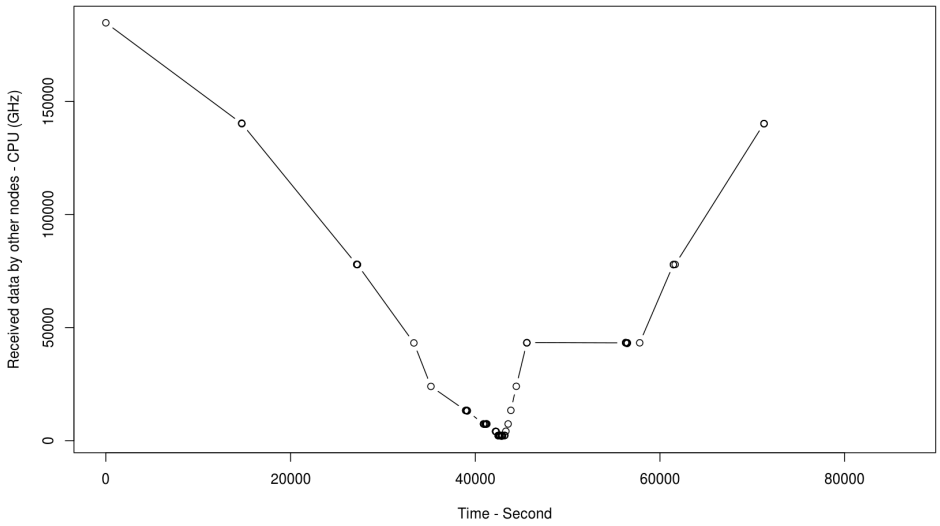

Figure F.9: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-05 and a growth factor f = 1.8.

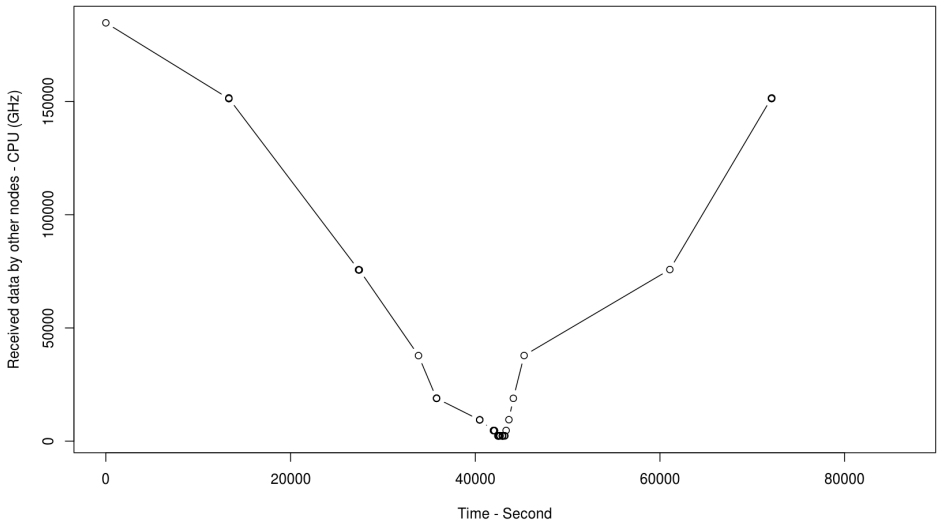

Figure F.10: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-05 and a growth factor f = 2.

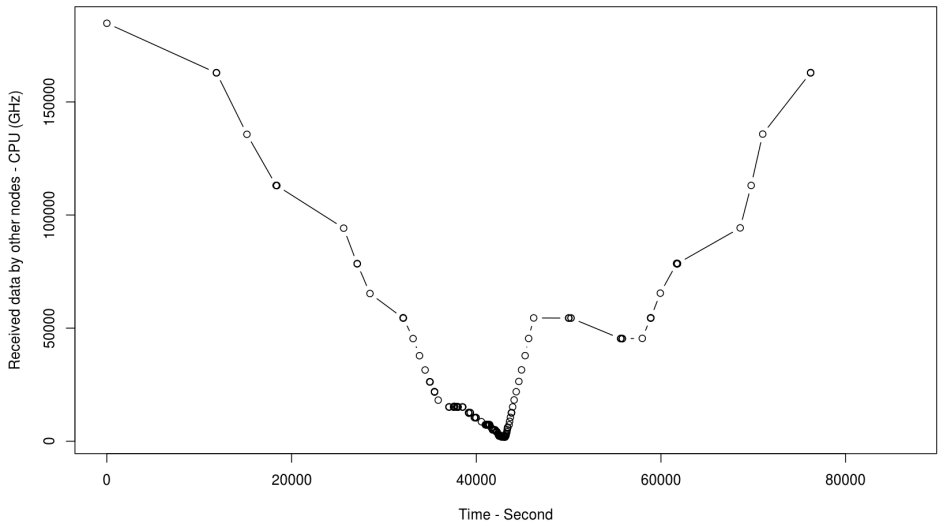

Figure F.11: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-04 and a growth factor f = 1.2.

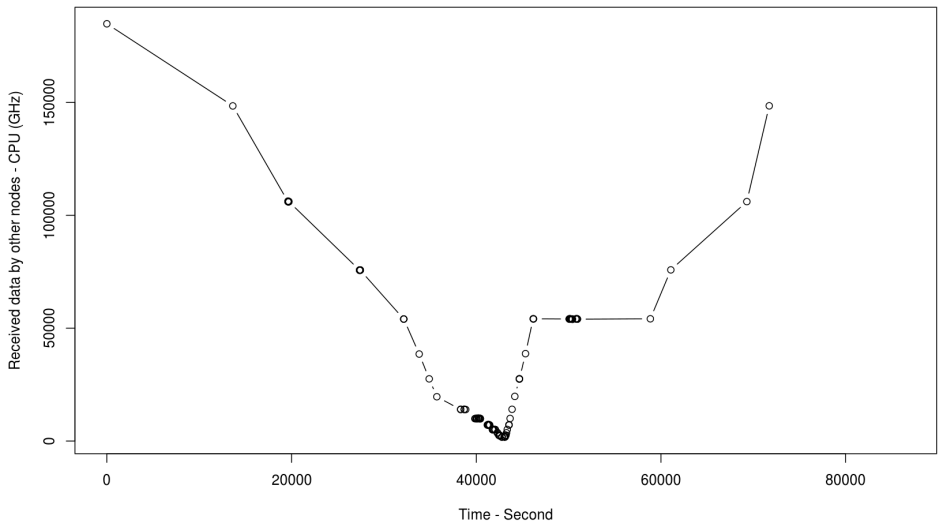

Figure F.12: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-04 and a growth factor f = 1.4.

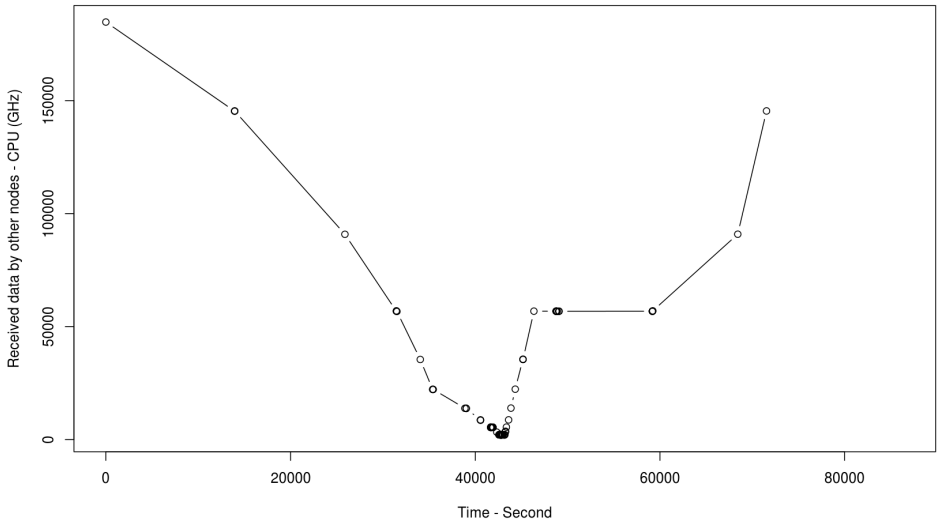

Figure F.13: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-04 and a growth factor f = 1.6.

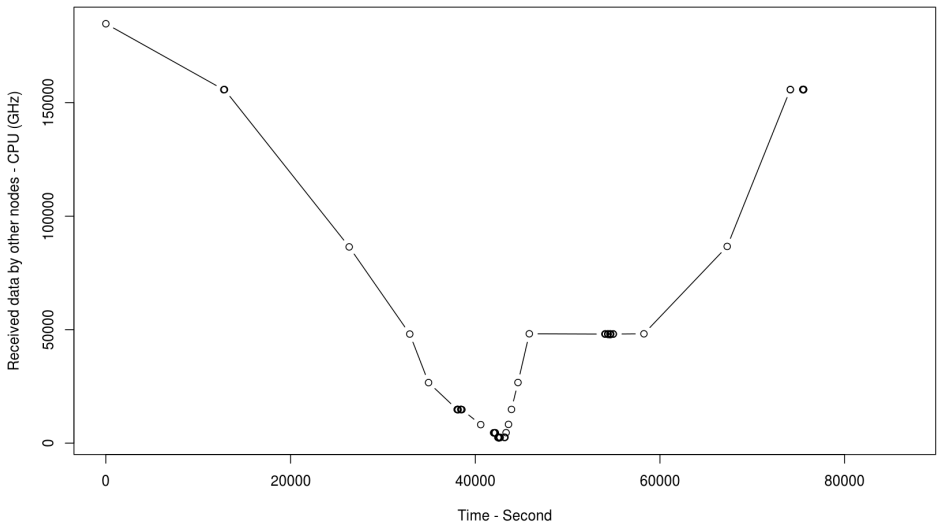

Figure F.14: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-04 and a growth factor f = 1.8.

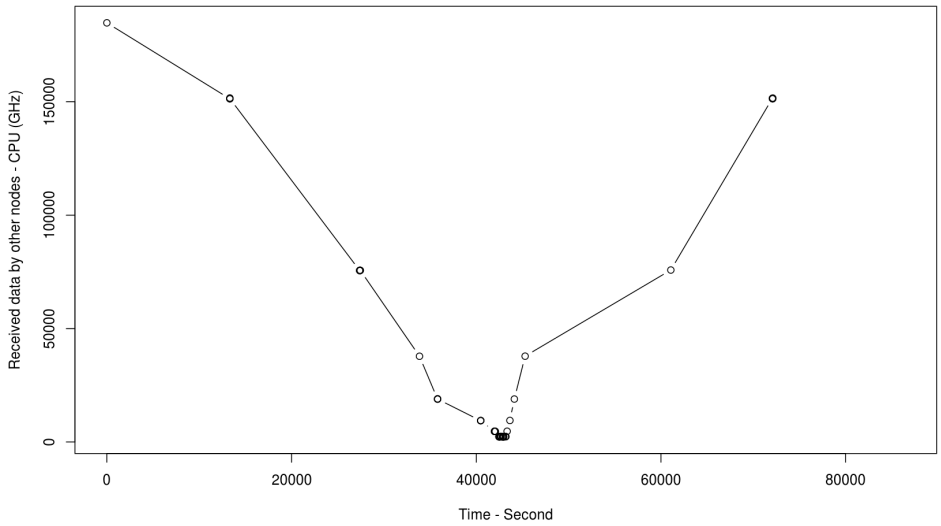

Figure F.15: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 1e-04 and a growth factor f = 2.

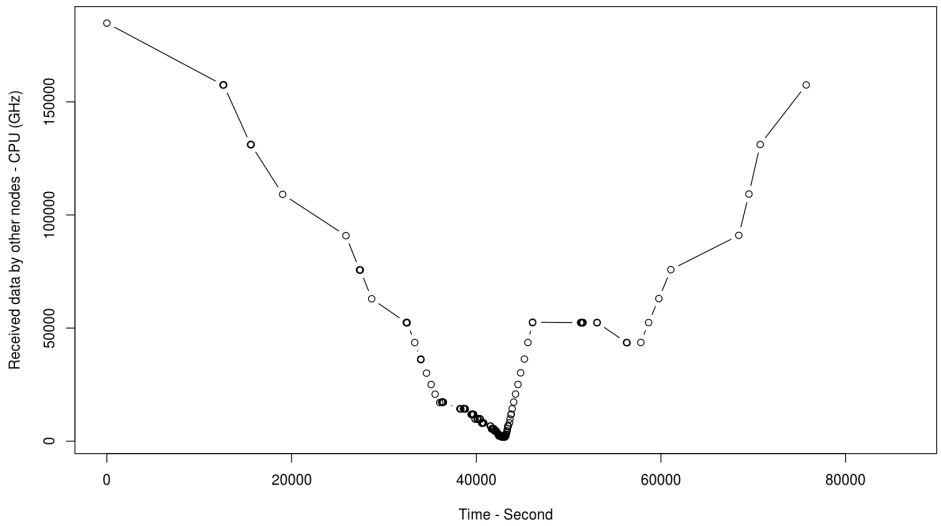

Figure F.16: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-04 and a growth factor f = 1.2.

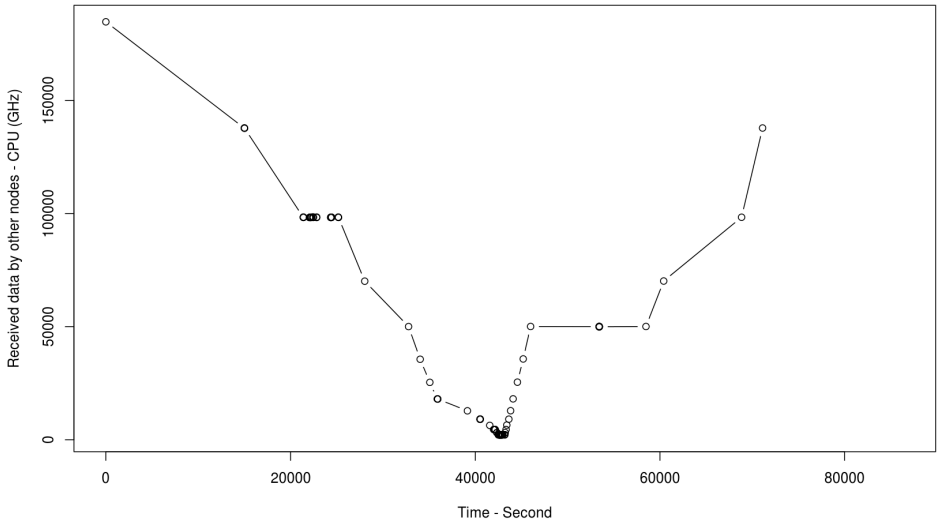

Figure F.17: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-04 and a growth factor f = 1.4.

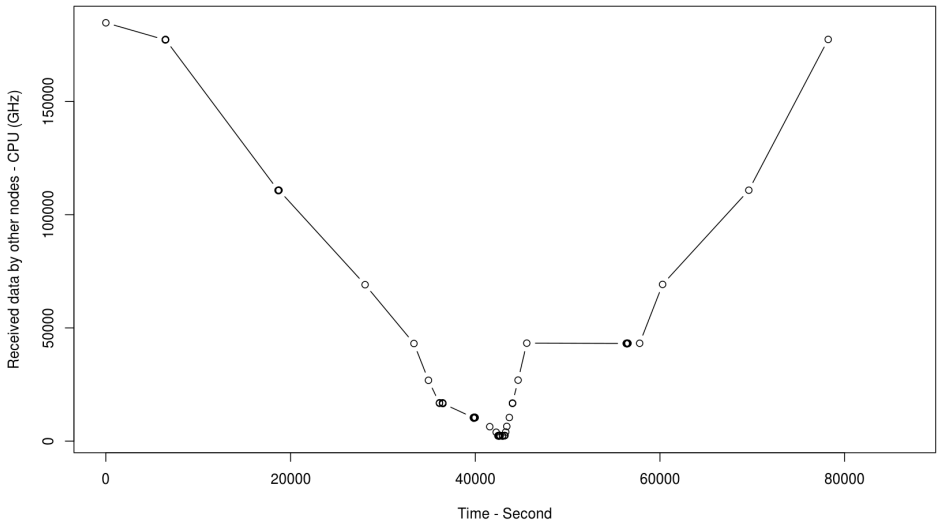

Figure F.18: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-04 and a growth factor f = 1.6.

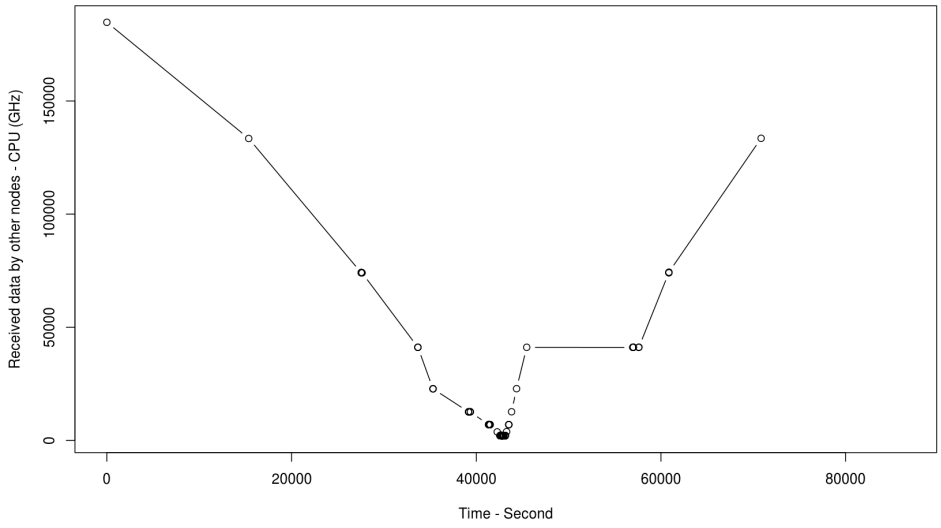

Figure F.19: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-04 and a growth factor f = 1.8.

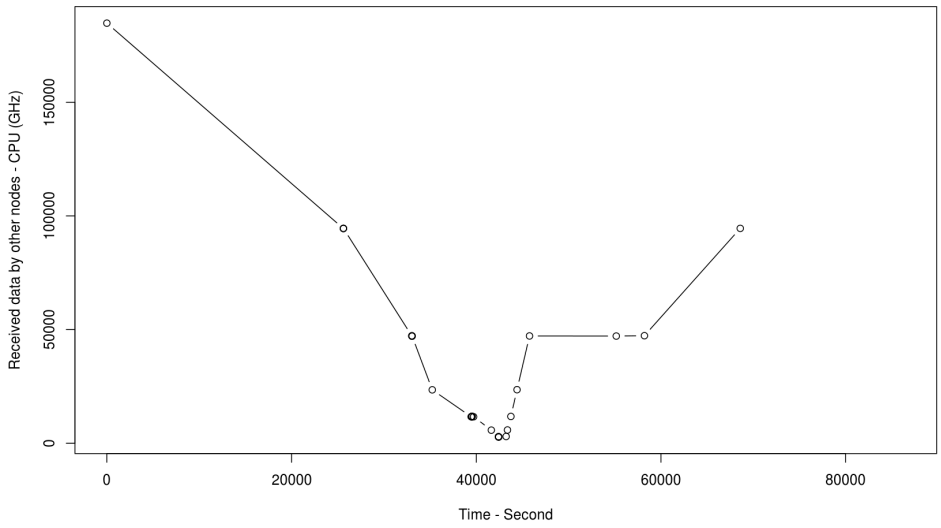

Figure F.20: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 5e-04 and a growth factor f = 2.

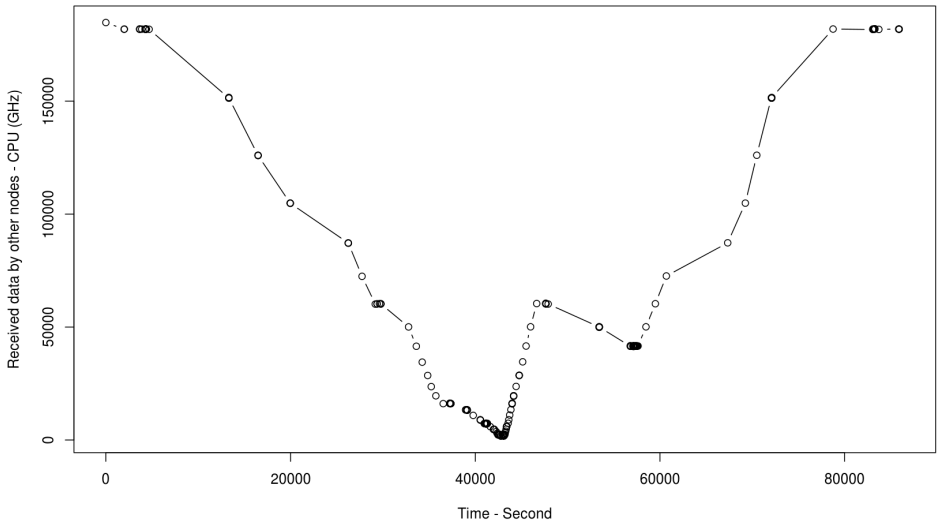

Figure F.21: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.001 and a growth factor f = 1.2.

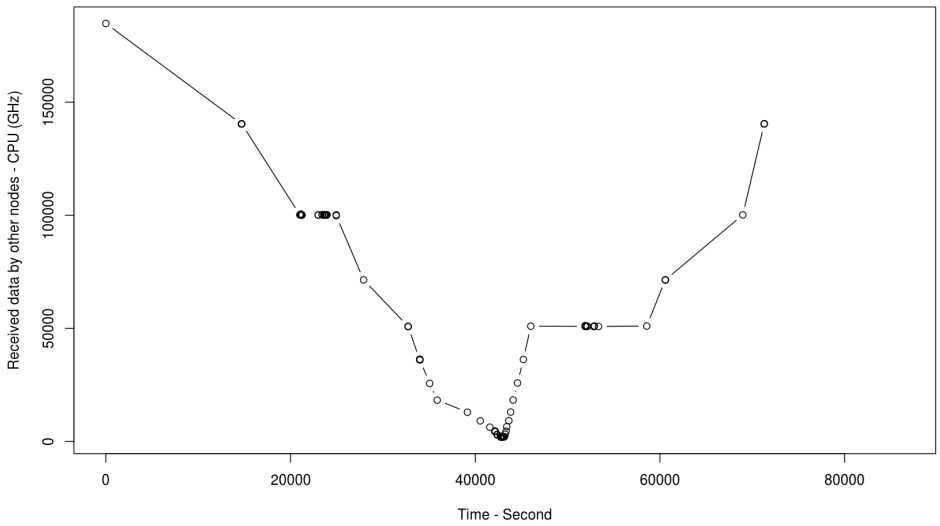

Figure F.22: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.001 and a growth factor f = 1.4.

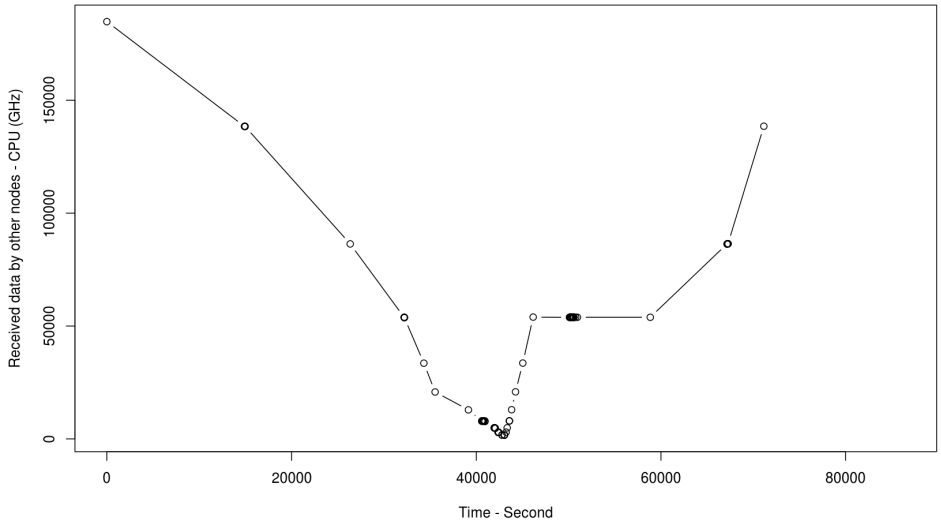

Figure F.23: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.001 and a growth factor f = 1.6.

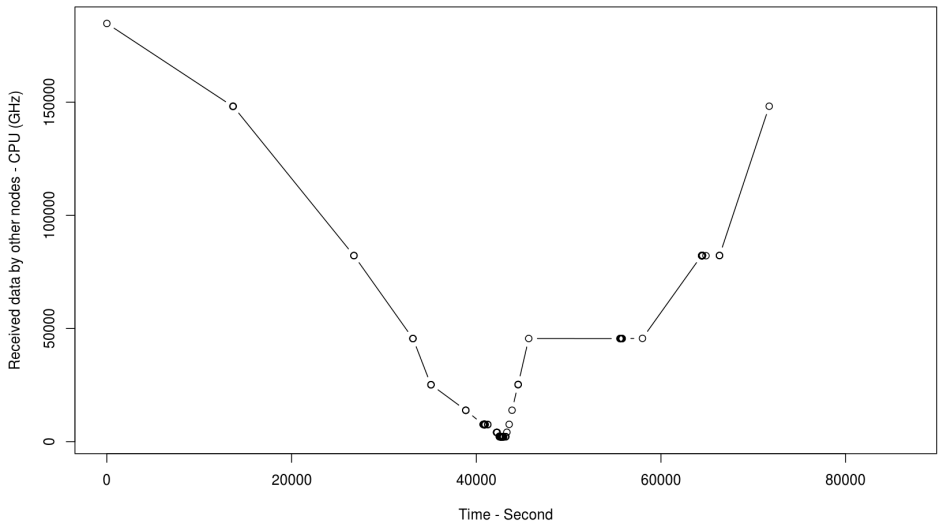

Figure F.24: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.001 and a growth factor f = 1.8.

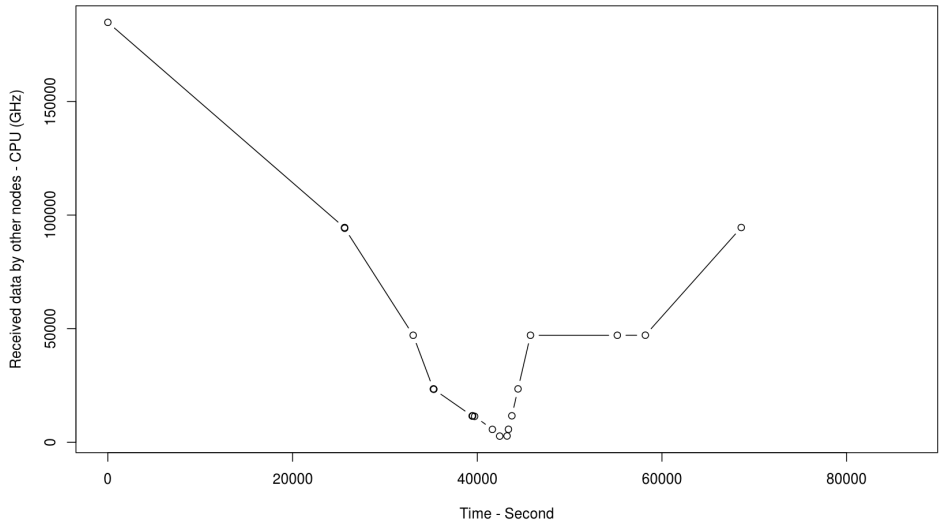

Figure F.25: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.001 and a growth factor f = 2.

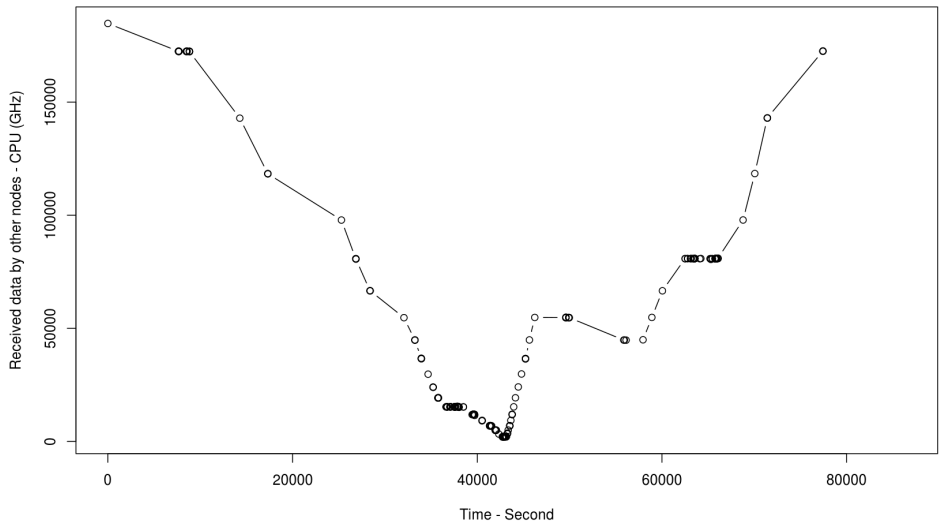

Figure F.26: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.005 and a growth factor f = 1.2.

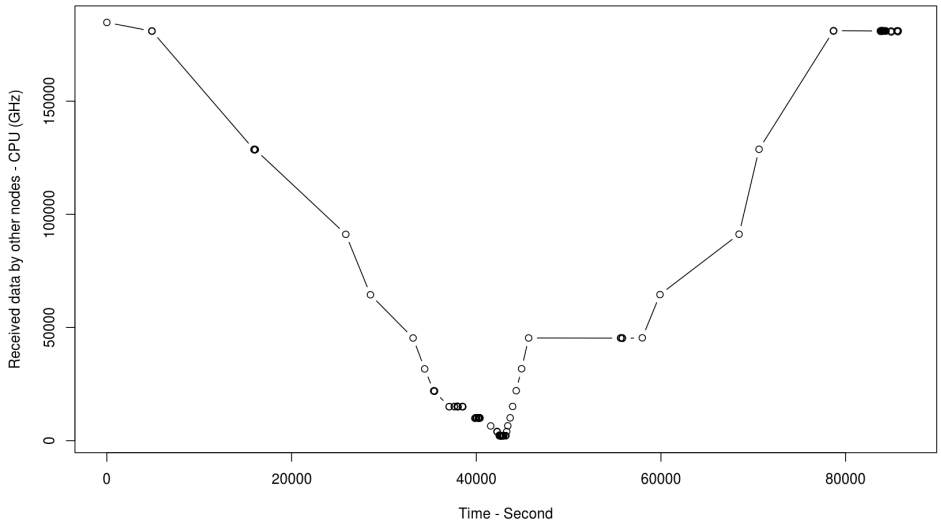

Figure F.27: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.005 and a growth factor f = 1.4.

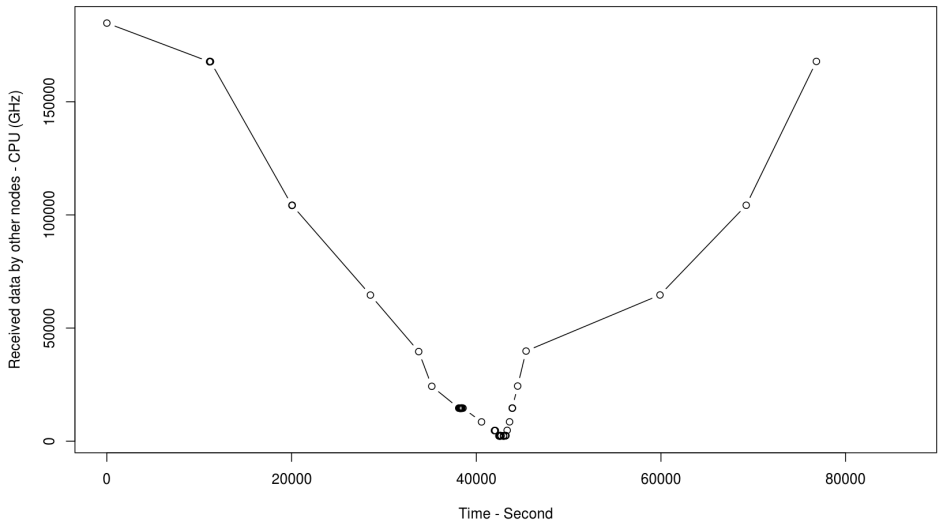

Figure F.28: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.005 and a growth factor f = 1.6.

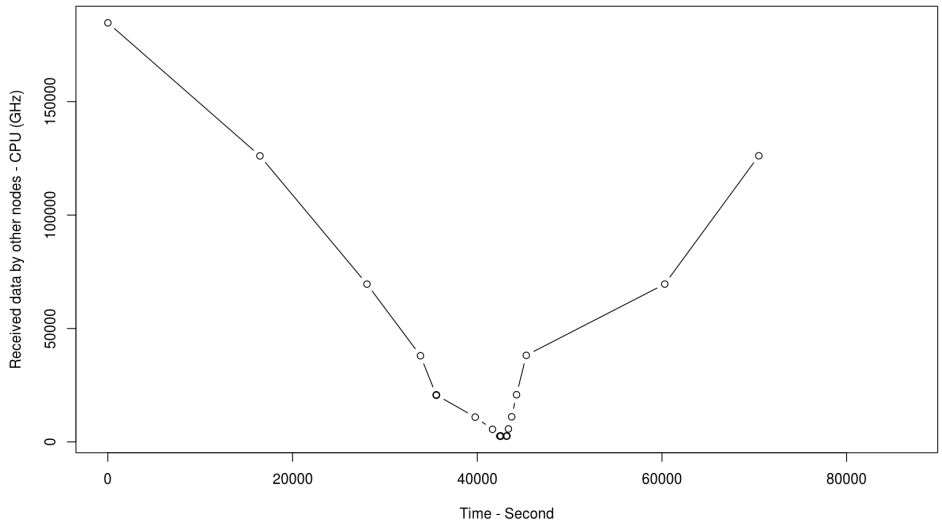

Figure F.29: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.005 and a growth factor f = 1.8.

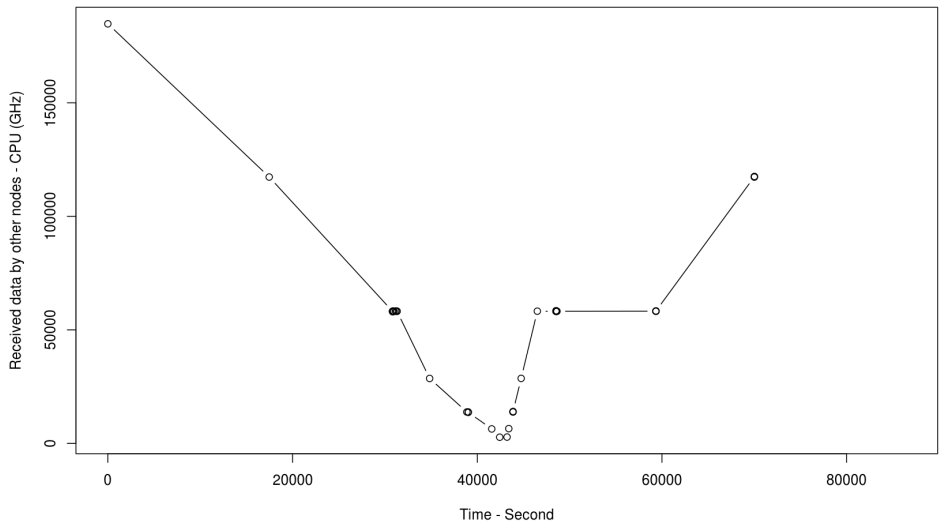

Figure F.30: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.005 and a growth factor f = 2.

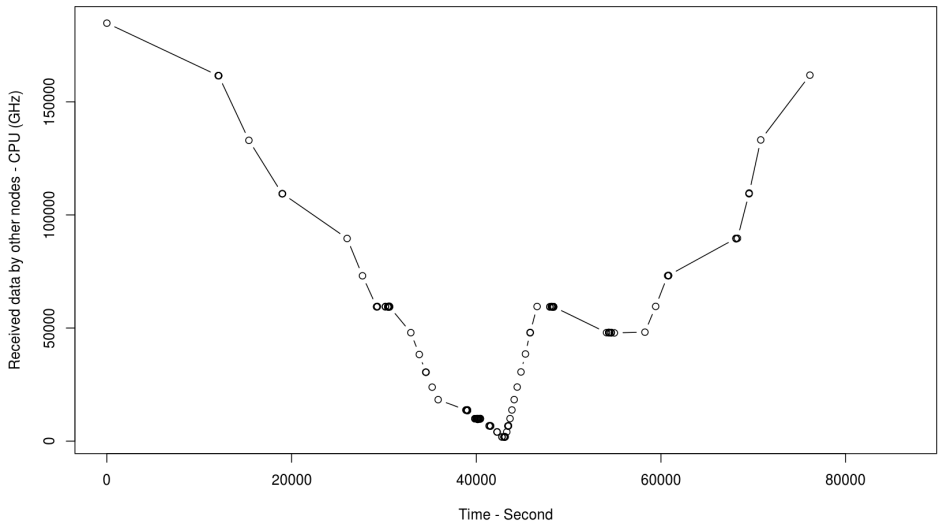

Figure F.31: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.01 and a growth factor f = 1.2.

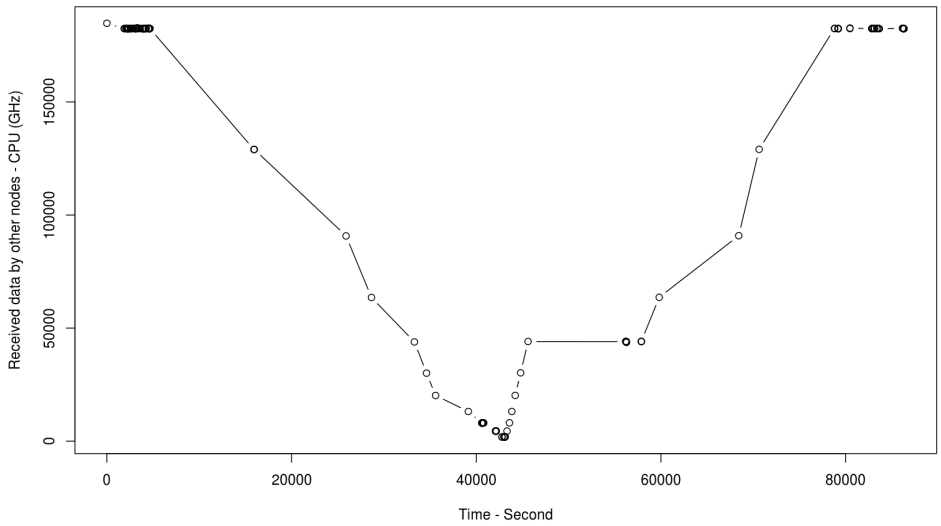

Figure F.32: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.01 and a growth factor f = 1.4.

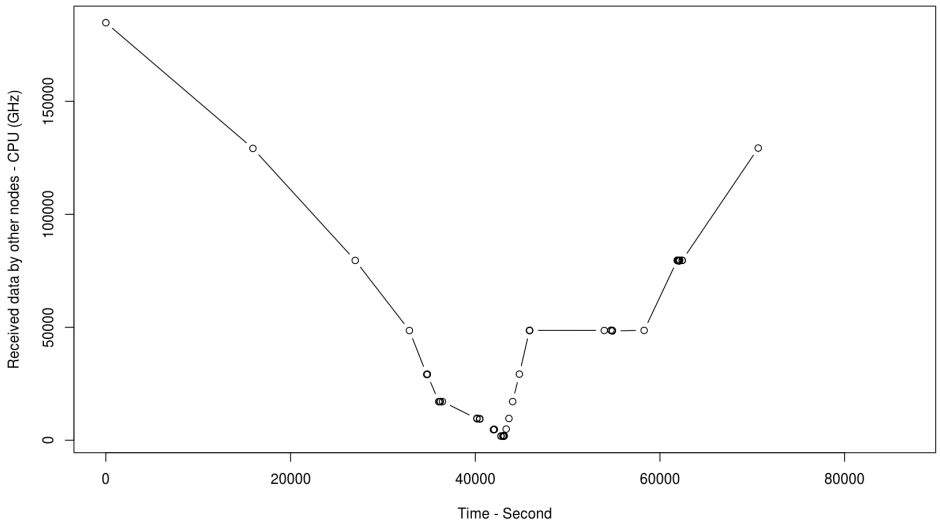

Figure F.33: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.01 and a growth factor f = 1.6.

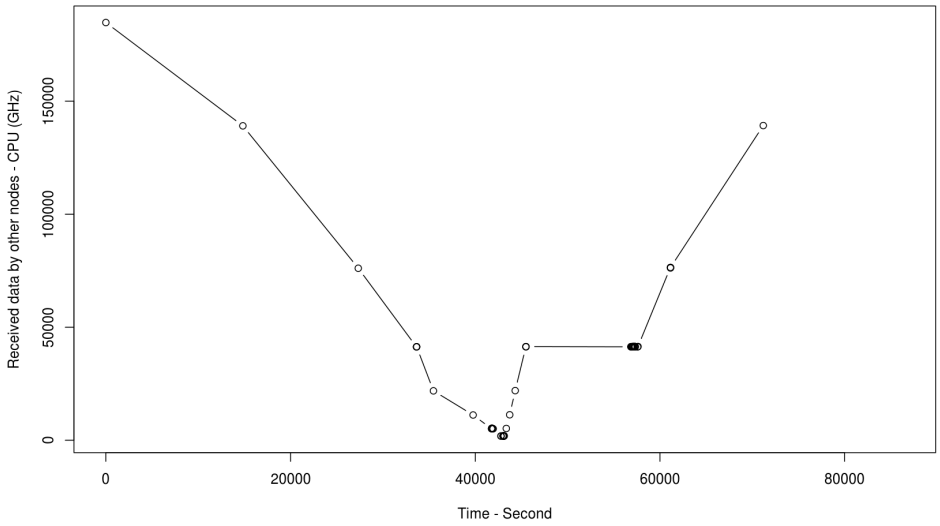

Figure F.34: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.01 and a growth factor f = 1.8.

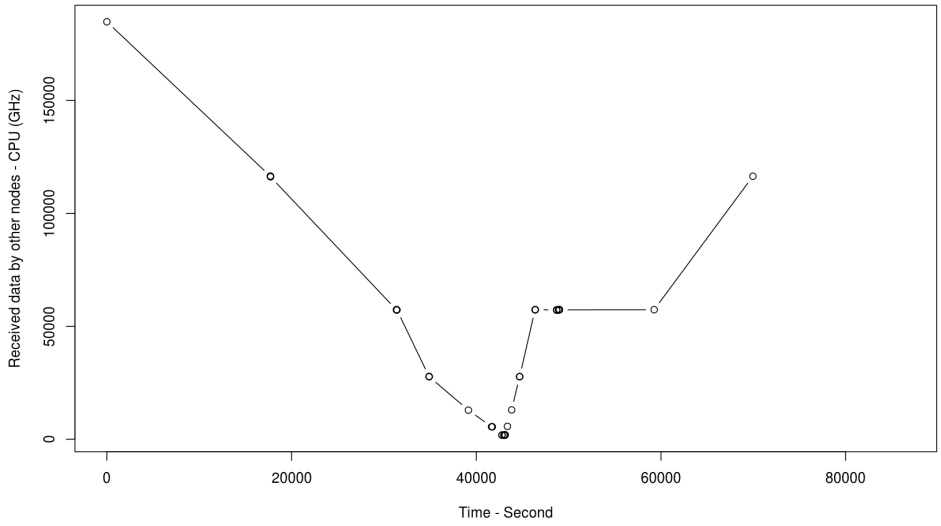

Figure F.35: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.01 and a growth factor $f = 2$.

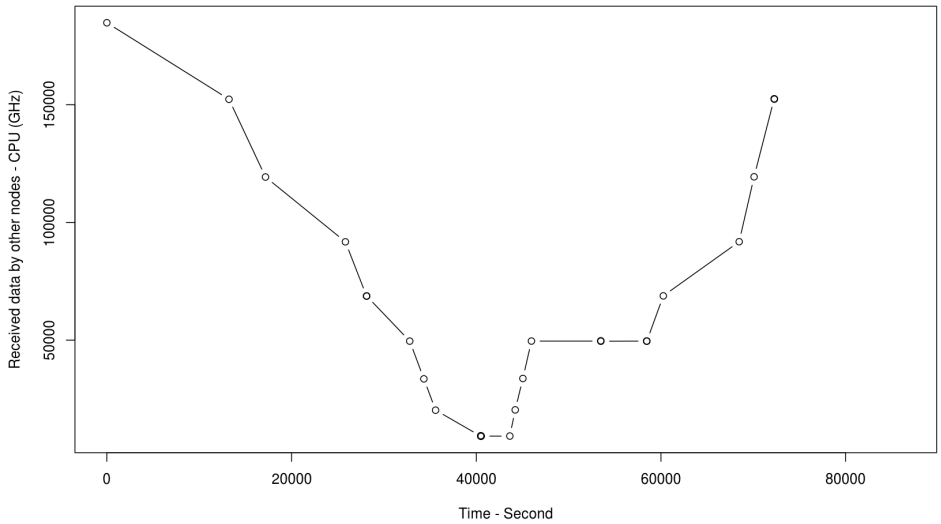

Figure F.36: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.05 and a growth factor $f = 1.2$.

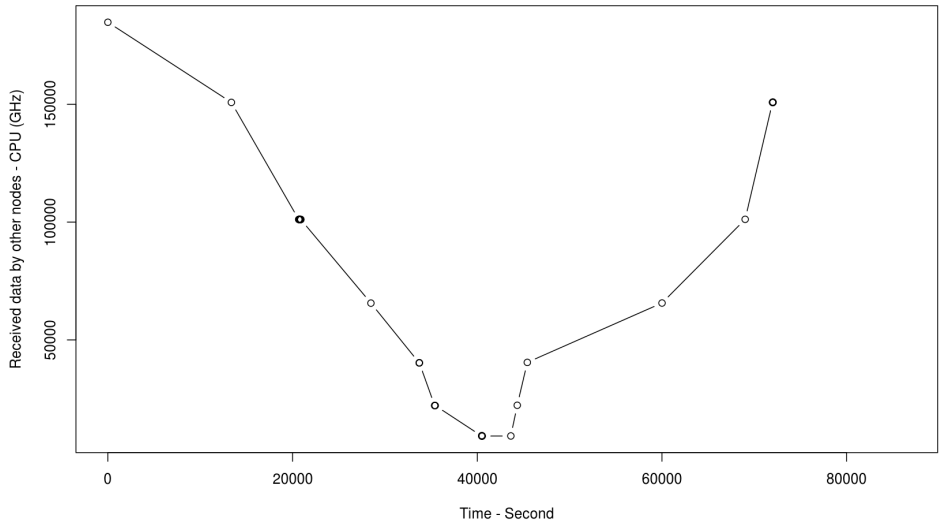

Figure F.37: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.05 and a growth factor f = 1.4.

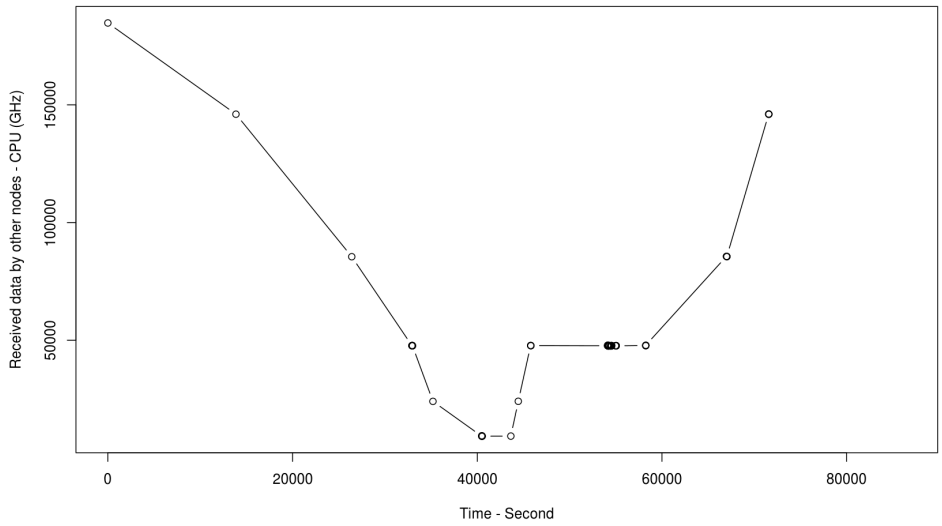

Figure F.38: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.05 and a growth factor f = 1.6.

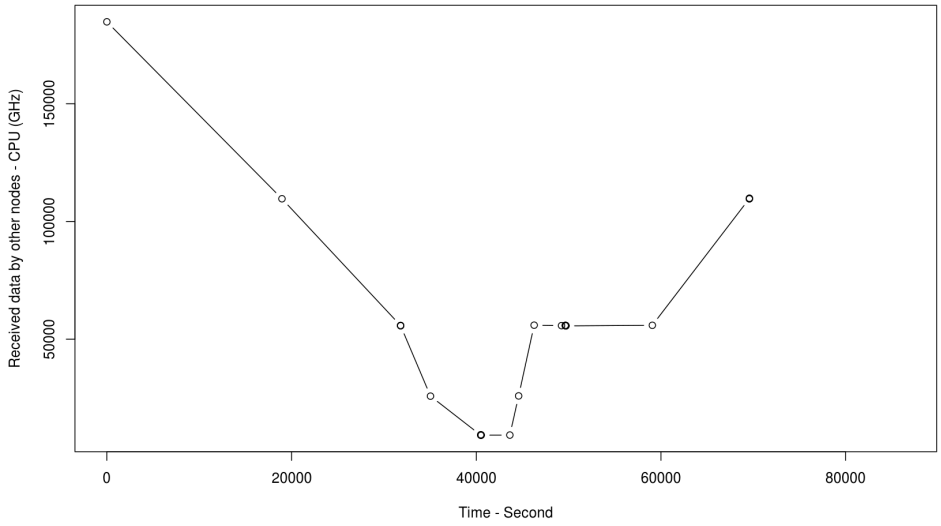

Figure F.39: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.05 and a growth factor f = 1.8.

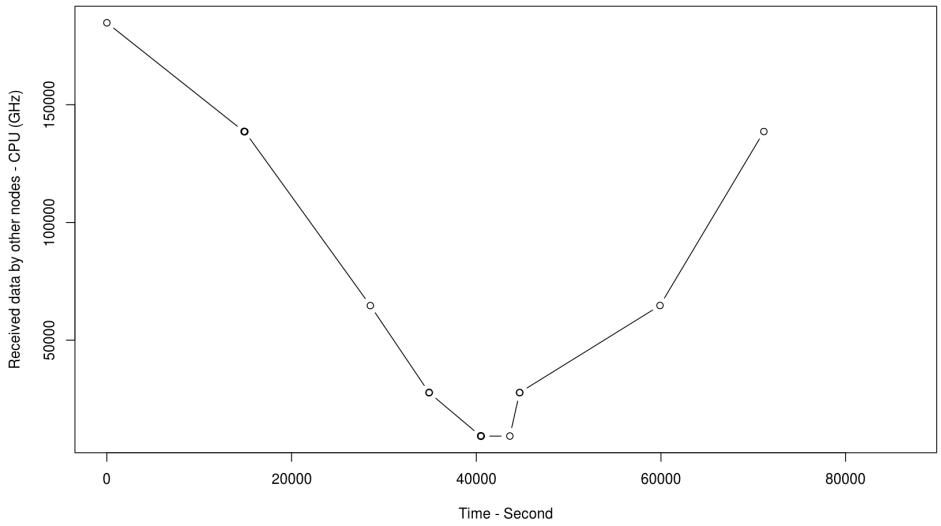

Figure F.40: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.05 and a growth factor f = 2.

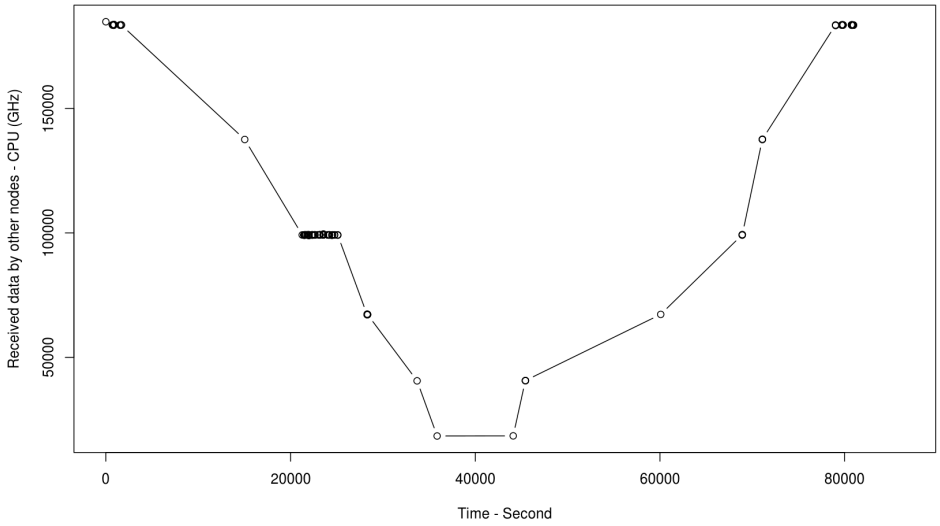

Figure F.41: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.1 and a growth factor f = 1.2.

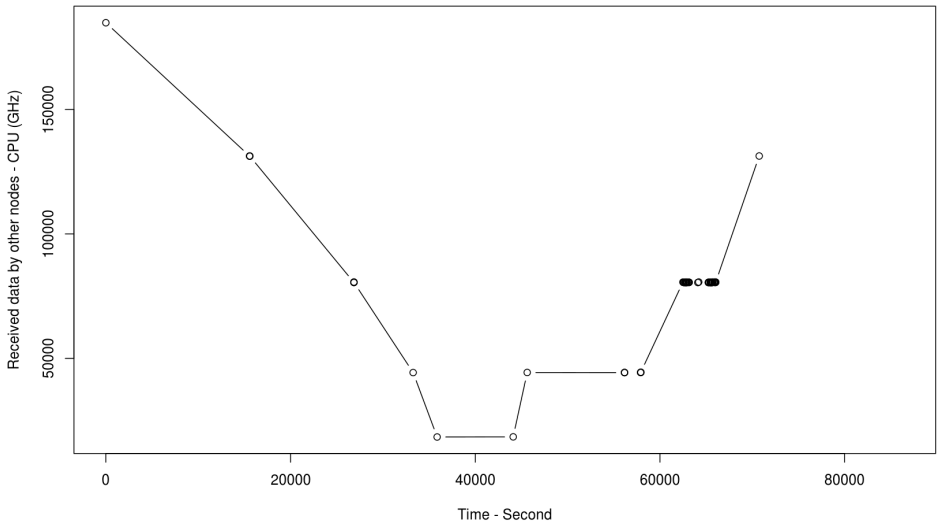

Figure F.42: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.1 and a growth factor f = 1.4.

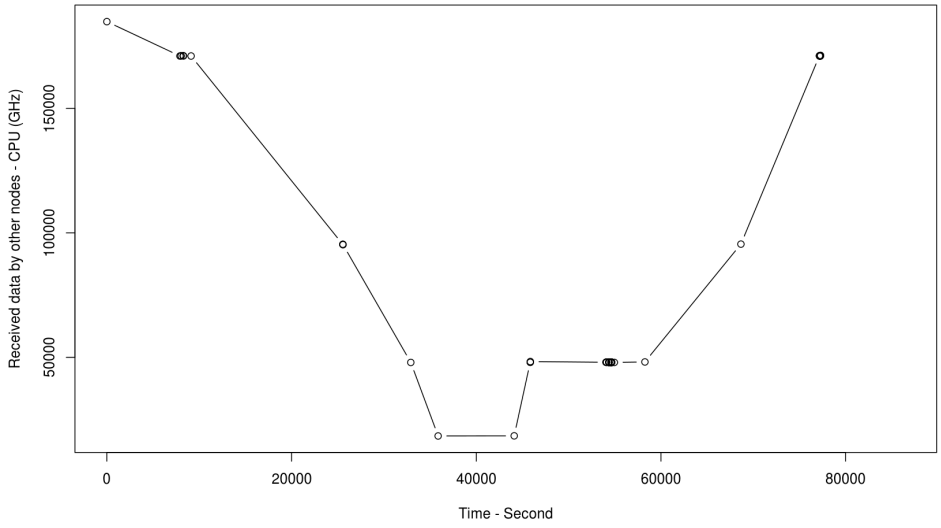

Figure F.43: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.1 and a growth factor f = 1.6.

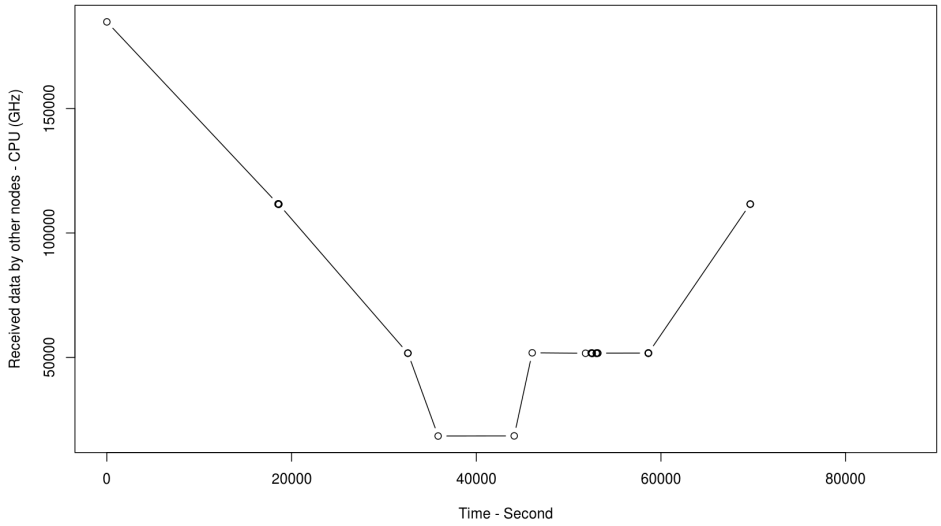

Figure F.44: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.1 and a growth factor f = 1.8.

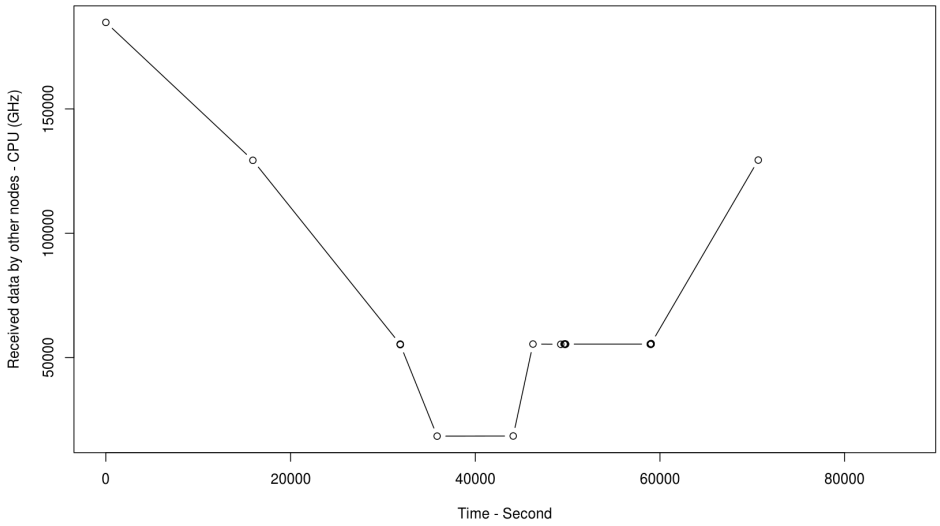

Figure F.45: Exponential-sized class-based update policy. How a cloud management system views data center's *CPU* capacity when a base factor β = 0.1 and a growth factor $f = 2$.

F.2 Data center's RAM

Table F.2: Exponential-sized class-based update policy. Number of Cloud LSA updates based on changes in a proposed data center's **RAM**. In this Table "β" refers to a *"base factor"*, "*f*" refers to *growth factor*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

β	*f*	**Number of updates**					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.00001	1.2	141	1	1	47	282	53.701
0.00001	1.4	79	1	1	17	207	34.596
0.00001	1.6	56	2	2	11	143	30.709
0.00001	1.8	45	2	2	9	147	26.189
0.00001	2	38	2	2	7	113	22.773
0.00005	1.2	141	1	1	49	310	57.426
0.00005	1.4	79	1	1	19	203	35.039
0.00005	1.6	52	2	2	15	148	28.071
0.00005	1.8	42	2	2	11	122	24.355
0.00005	2	40	2	2	7	141	24.408
0.0001	1.2	140	1	1	53	287	51.221
0.0001	1.4	84	1	1	17	215	39.193
0.0001	1.6	55	2	2	11	129	27.770
0.0001	1.8	43	2	2	11	141	28.146
0.0001	2	40	2	2	7	139	24.852
0.0005	1.2	145	1	1	61	325	52.074
0.0005	1.4	73	1	1	11	157	31.439

Continued on next page

Table F.2 – continued from previous page

β	f	Number of updates					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.0005	1.6	73	2	2	19	186	37.119
0.0005	1.8	42	2	2	13	113	21.012
0.0005	2	31	2	2	5	121	23.444
0.001	1.2	178	1	1	65	329	50.323
0.001	1.4	70	1	1	23	156	29.052
0.001	1.6	49	2	2	13	113	25.803
0.001	1.8	40	2	2	9	112	23.161
0.001	2	30	2	2	7	125	22.715
0.005	1.2	120	1	1	57	268	35.726
0.005	1.4	106	1	1	27	209	41.497
0.005	1.6	59	2	2	15	160	25.509
0.005	1.8	42	2	2	9	95	19.383
0.005	2	37	2	2	9	143	20.035
0.01	1.2	108	1	1	41	218	35.398
0.01	1.4	113	1	1	36	267	47.877
0.01	1.6	42	2	2	11	123	19.939
0.01	1.8	38	2	2	9	115	18.625
0.01	2	36	2	2	9	93	18.086
0.05	1.2	65	1	1	17	147	25.066
0.05	1.4	49	1	1	13	111	23.715
0.05	1.6	32	2	2	11	109	21.170
0.05	1.8	35	2	2	7	139	22.192
0.05	2	32	2	2	7	123	18.515

Continued on next page

Table F.2 – continued from previous page

β	f	Number of updates					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.1	1.2	114	1	1	22	290	49.928
0.1	1.4	25	1	1	9	85	15.414
0.1	1.6	31	2	2	7	127	18.413
0.1	1.8	29	2	2	5	87	17.987
0.1	2	25	2	2	5	95	17.794

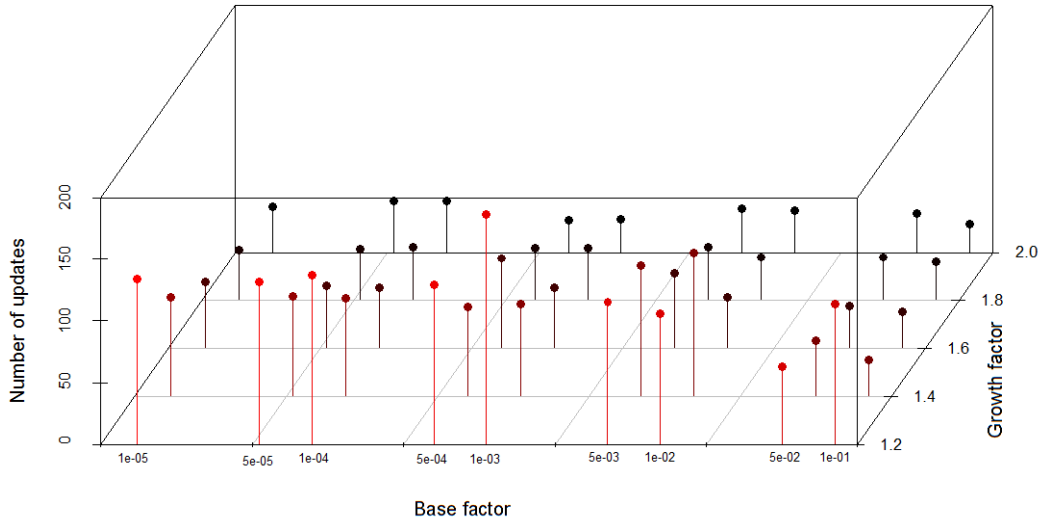

Figure F.46: Exponential-sized class-based update policy. Number of updates per different growth factor and base factor values based on changes of a data center's *RAM* capacity. The "X" axis is in logarithm scale.

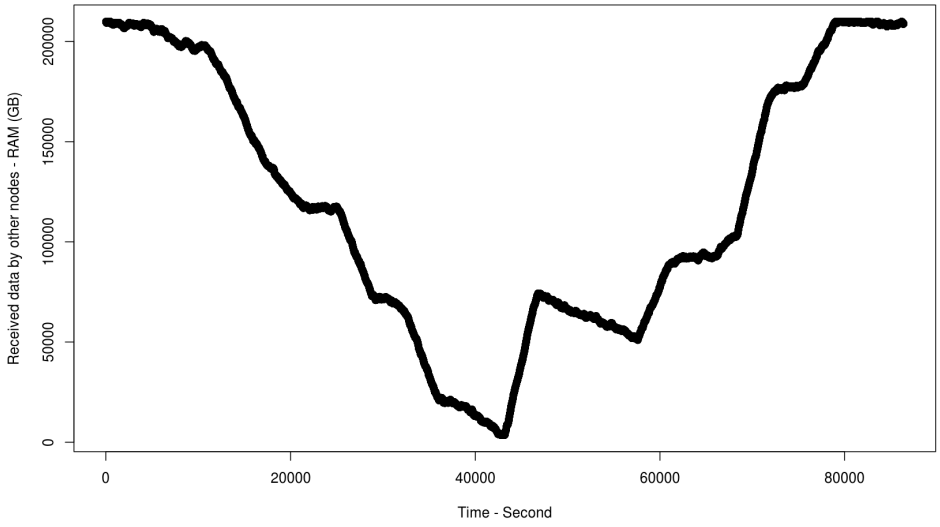

Figure F.47: Data center sample *RAM* (GB) capacity per second.

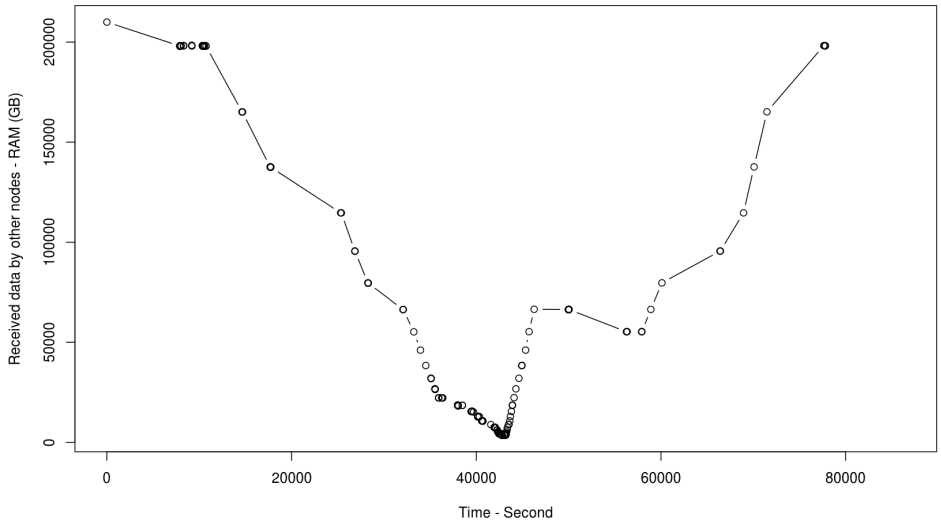

Figure F.48: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-05 and a growth factor f = 1.2.

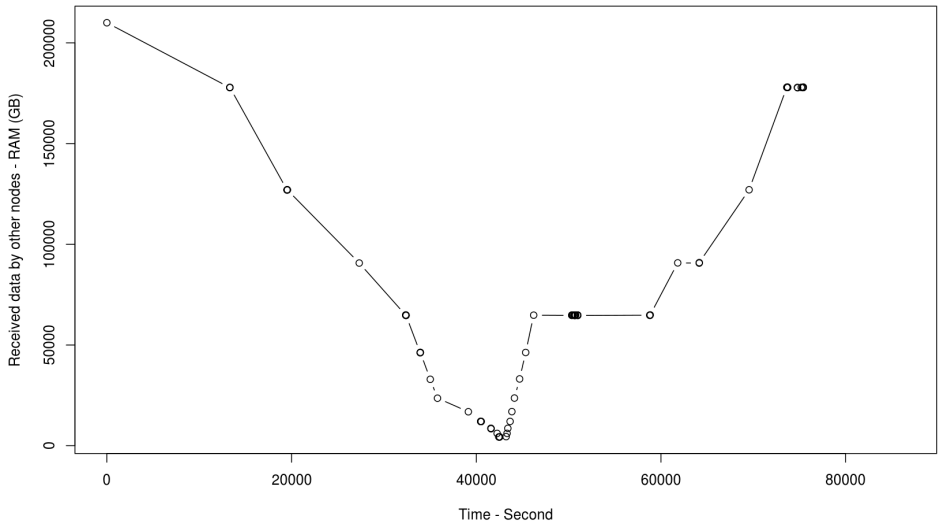

Figure F.49: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-05 and a growth factor f = 1.4.

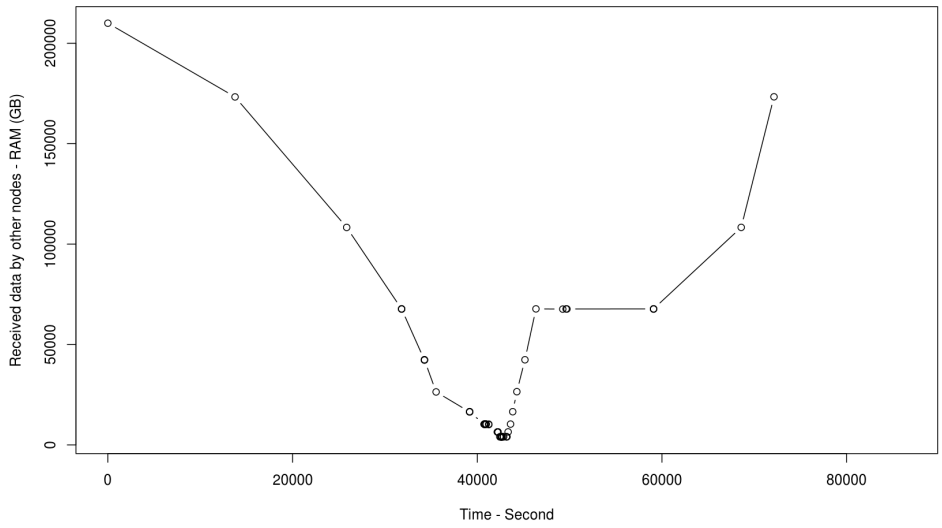

Figure F.50: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-05 and a growth factor f = 1.6.

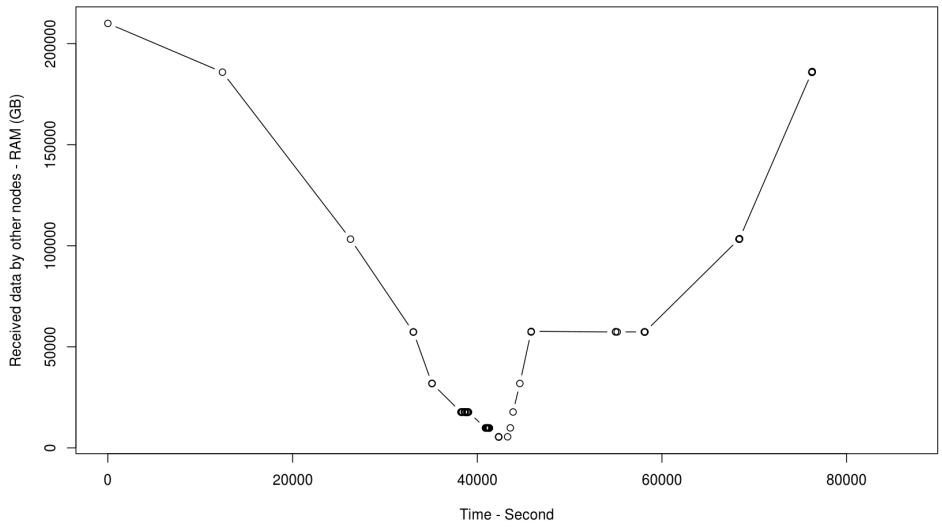

Figure F.51: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-05 and a growth factor f = 1.8.

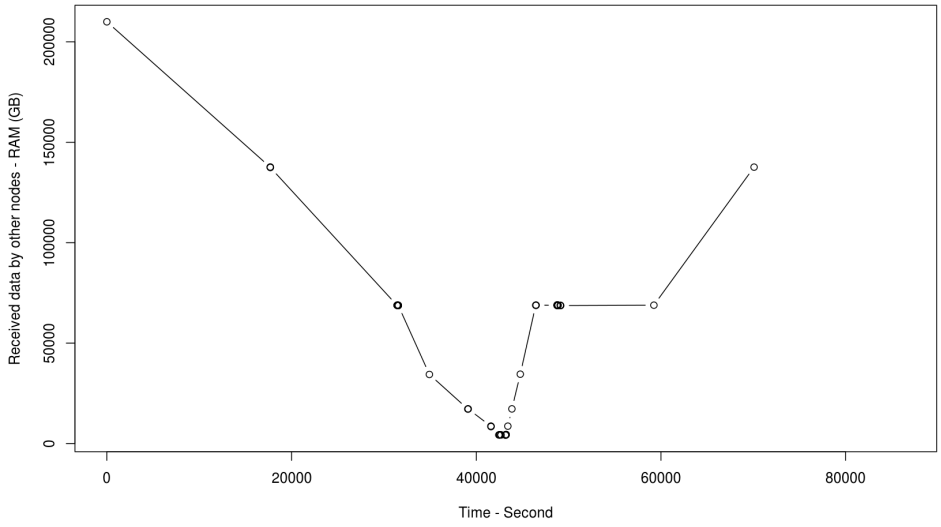

Figure F.52: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-05 and a growth factor f = 2.

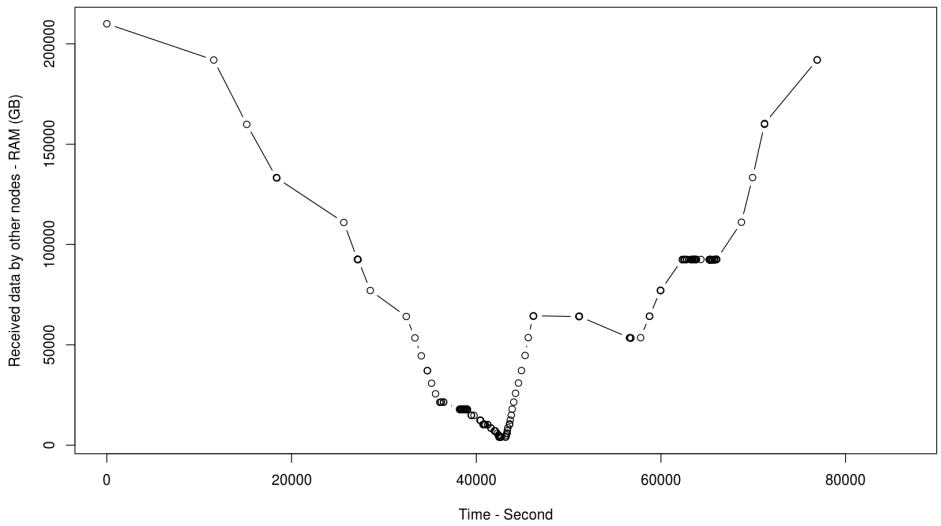

Figure F.53: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-05 and a growth factor f = 1.2.

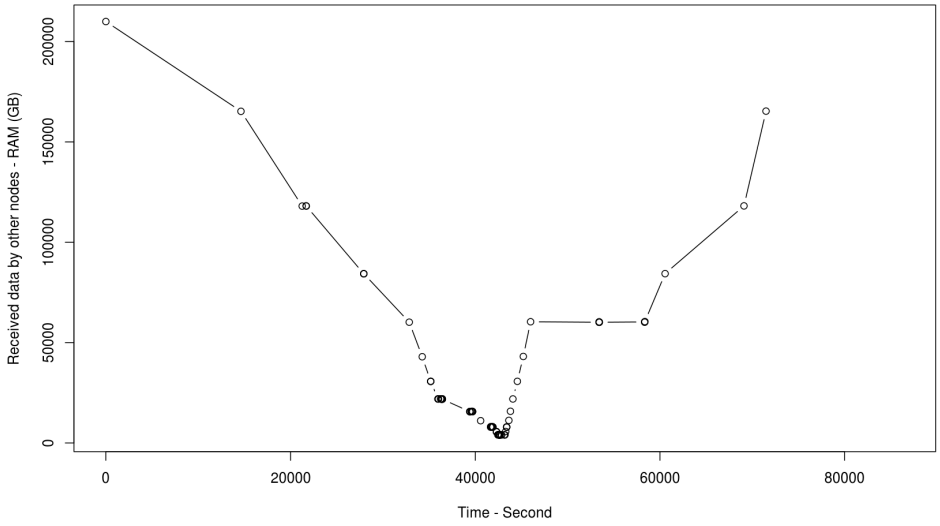

Figure F.54: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-05 and a growth factor f = 1.4.

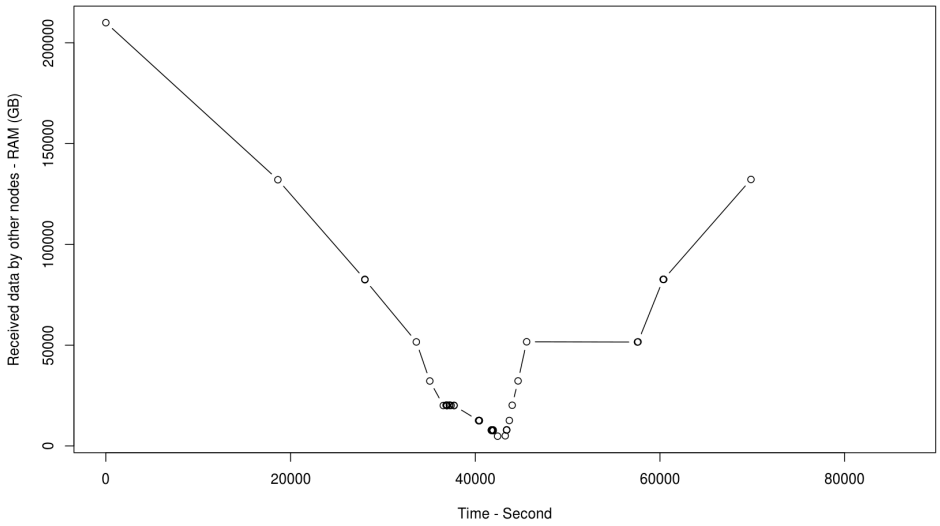

Figure F.55: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-05 and a growth factor f = 1.6.

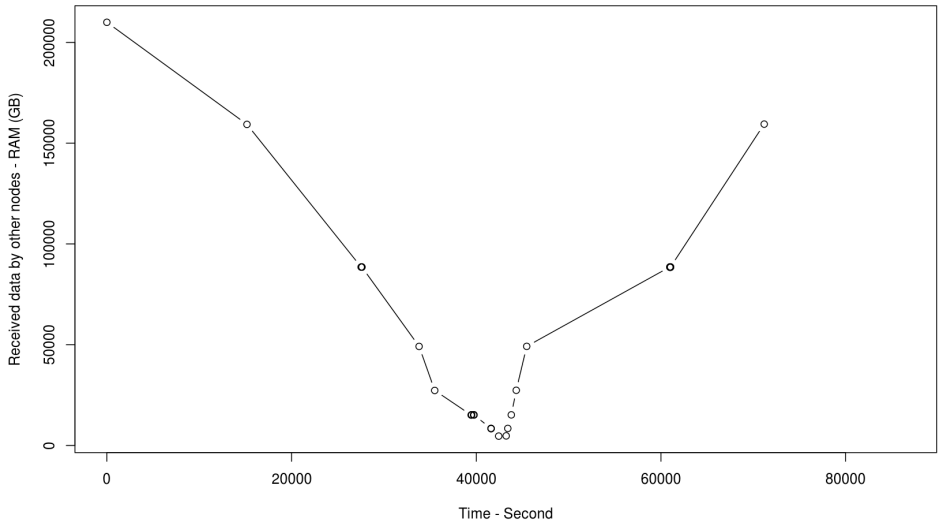

Figure F.56: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-05 and a growth factor f = 1.8.

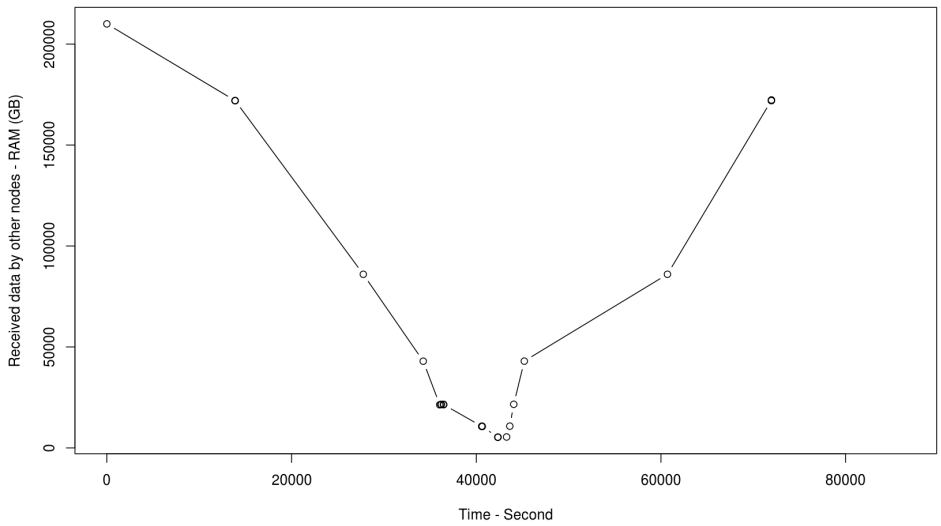

Figure F.57: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-05 and a growth factor f = 2.

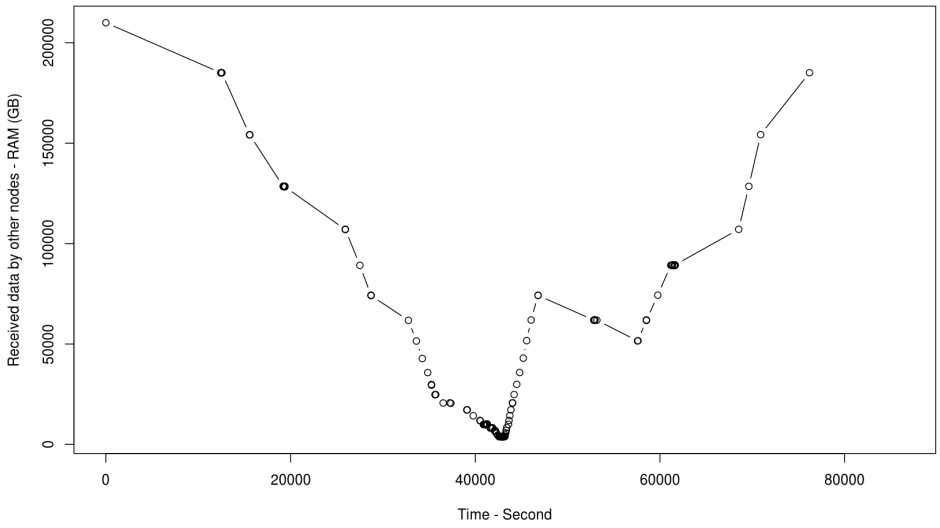

Figure F.58: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-04 and a growth factor f = 1.2.

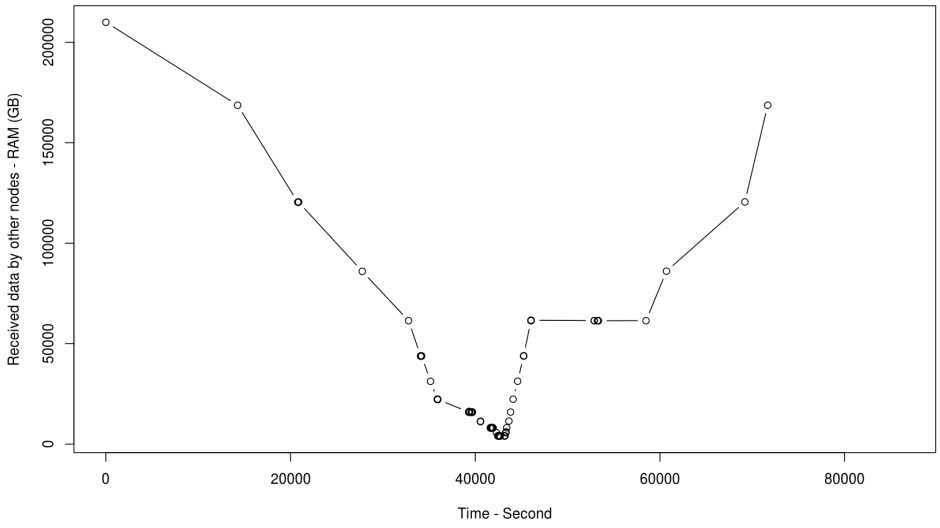

Figure F.59: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-04 and a growth factor f = 1.4.

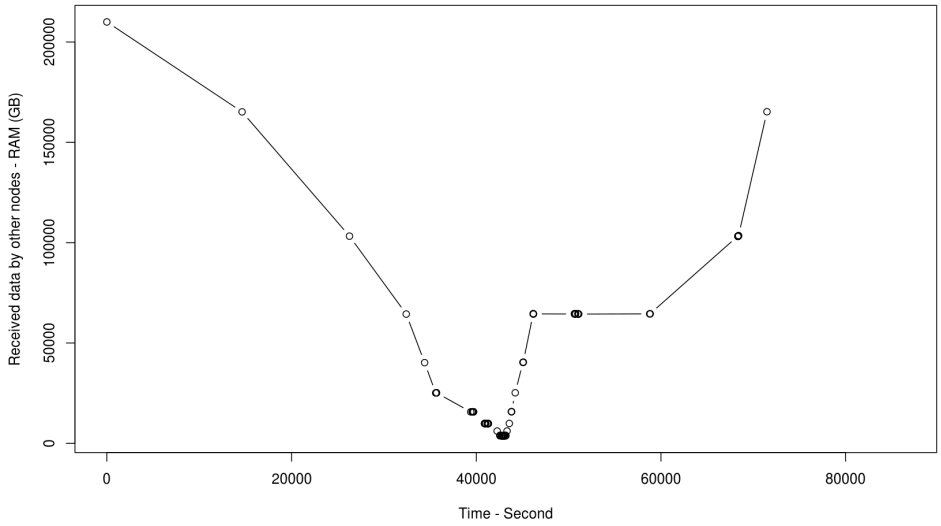

Figure F.60: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-04 and a growth factor f = 1.6.

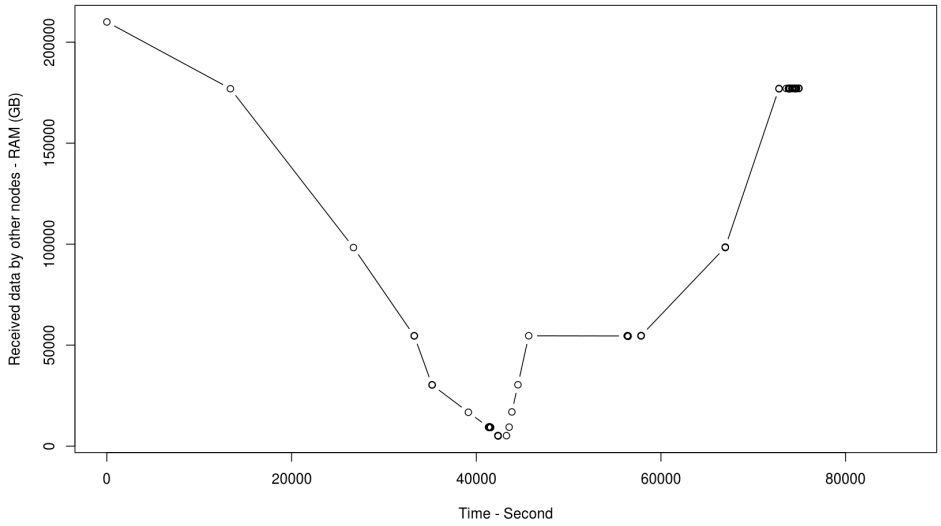

Figure F.61: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-04 and a growth factor f = 1.8.

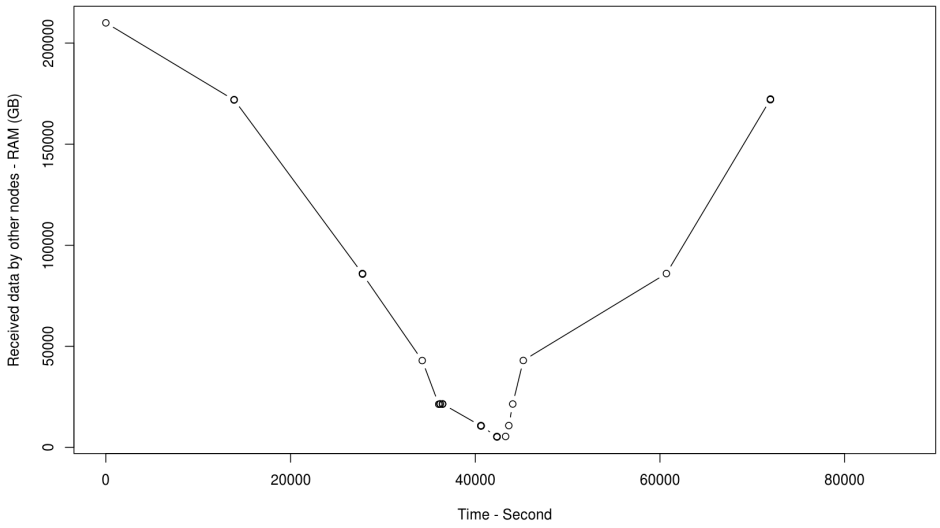

Figure F.62: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 1e-04 and a growth factor f = 2.

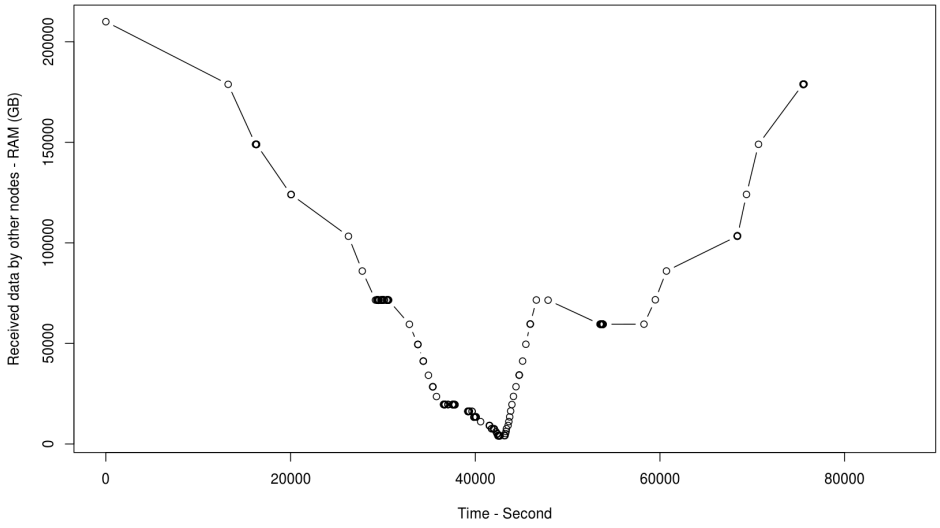

Figure F.63: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-04 and a growth factor f = 1.2.

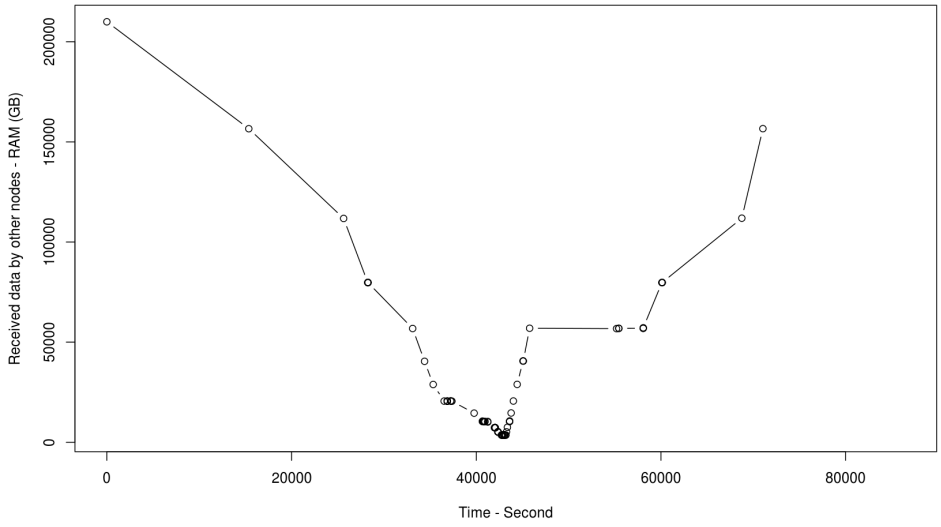

Figure F.64: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-04 and a growth factor f = 1.4.

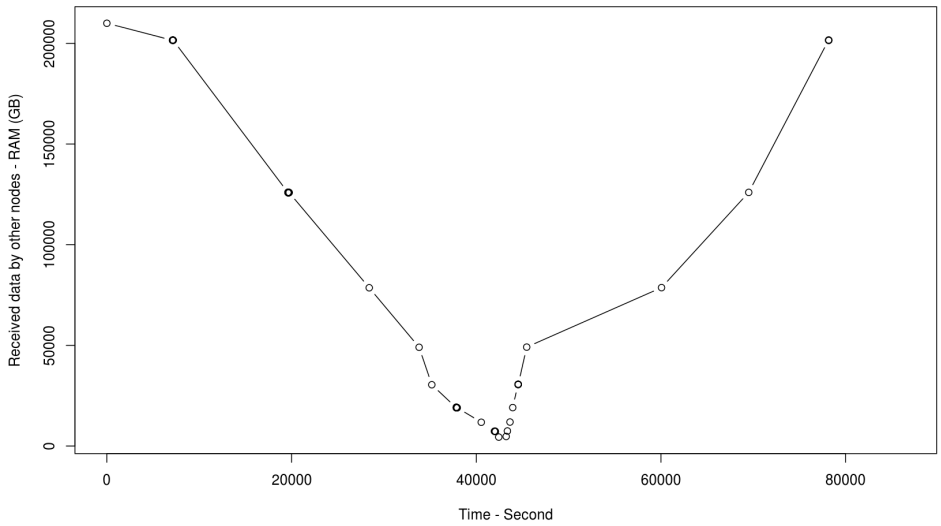

Figure F.65: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-04 and a growth factor f = 1.6.

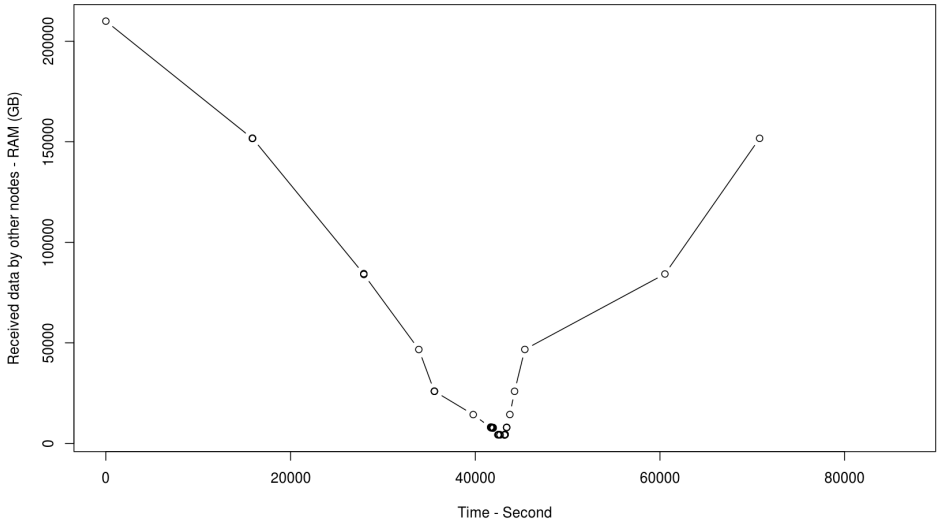

Figure F.66: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-04 and a growth factor f = 1.8.

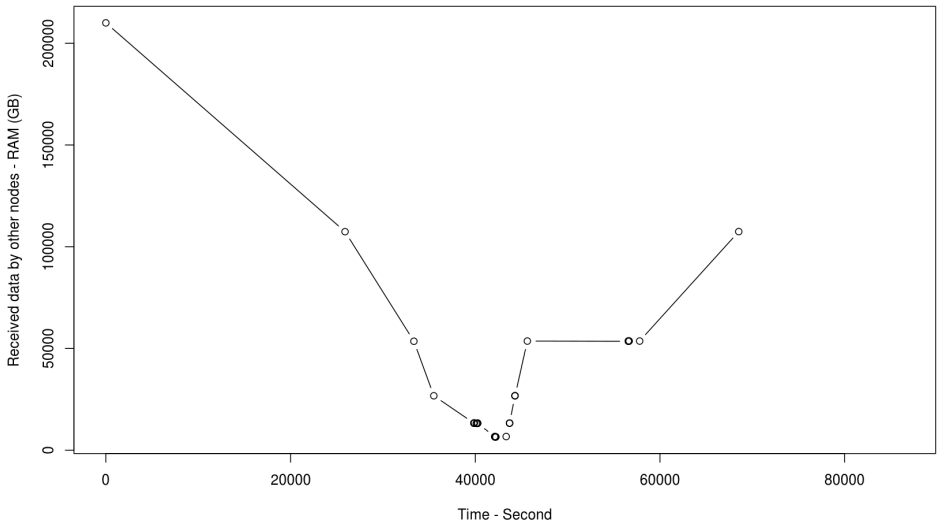

Figure F.67: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 5e-04 and a growth factor f = 2.

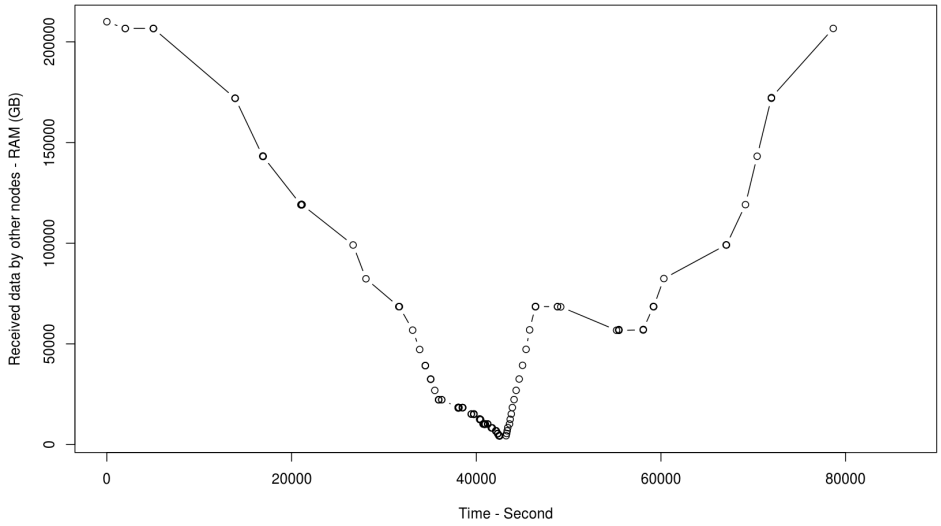

Figure F.68: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.001 and a growth factor f = 1.2.

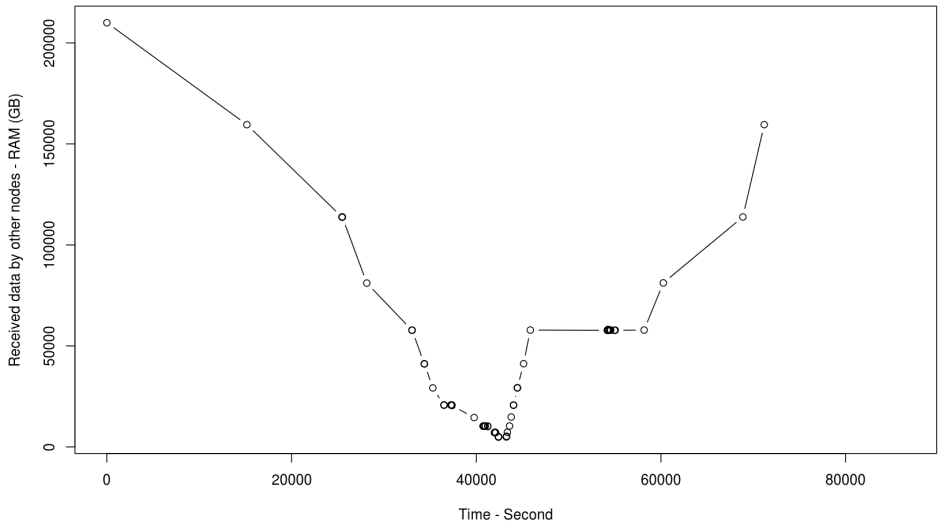

Figure F.69: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.001 and a growth factor f = 1.4.

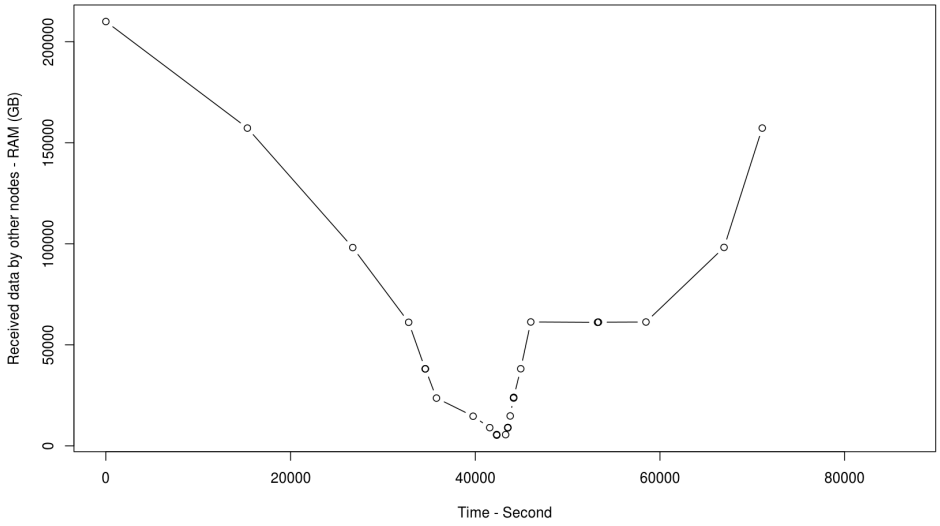

Figure F.70: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.001 and a growth factor f = 1.6.

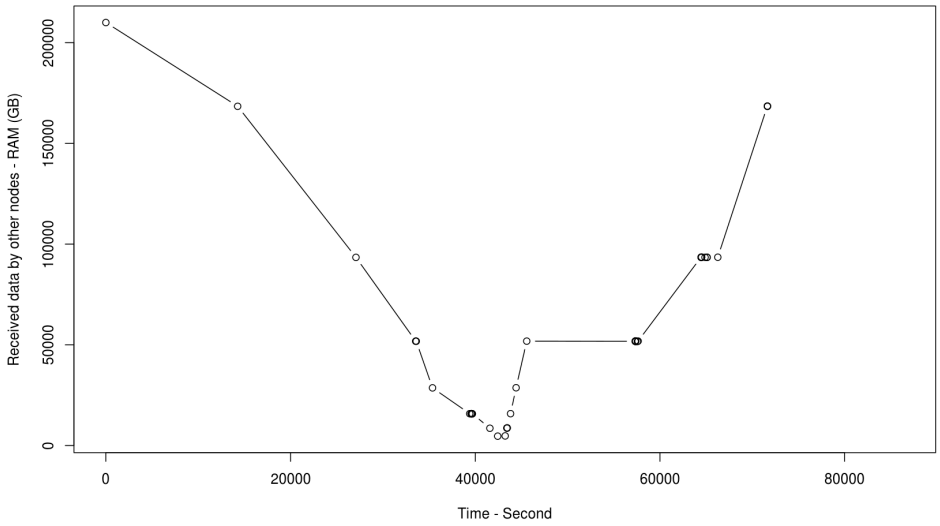

Figure F.71: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.001 and a growth factor f = 1.8.

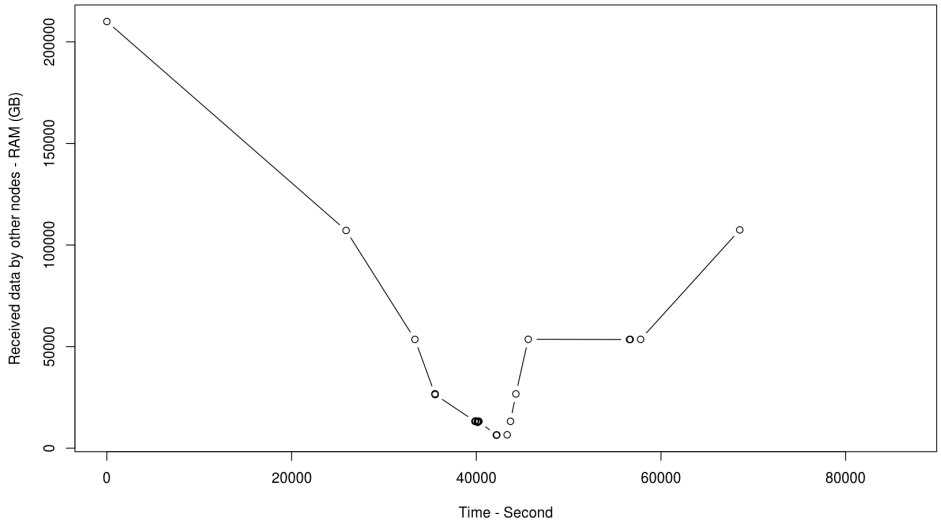

Figure F.72: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.001 and a growth factor f = 2.

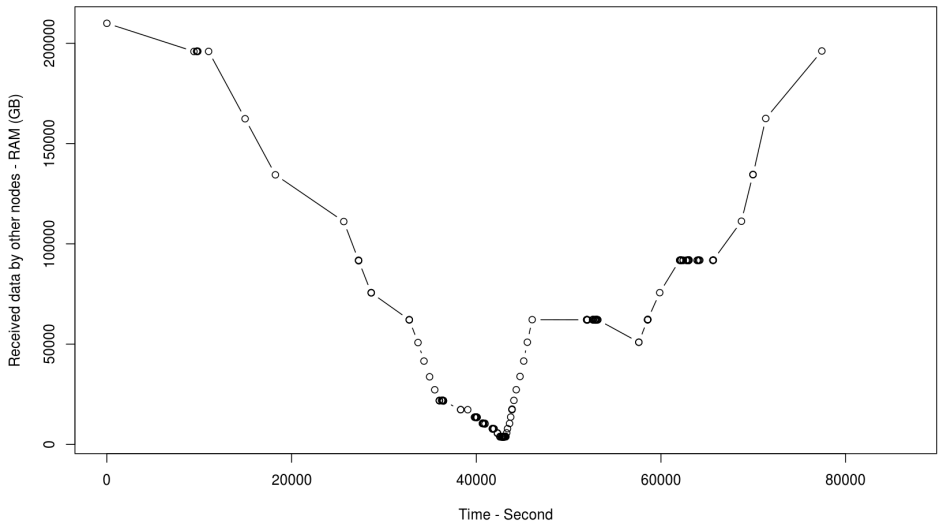

Figure F.73: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.005 and a growth factor f = 1.2.

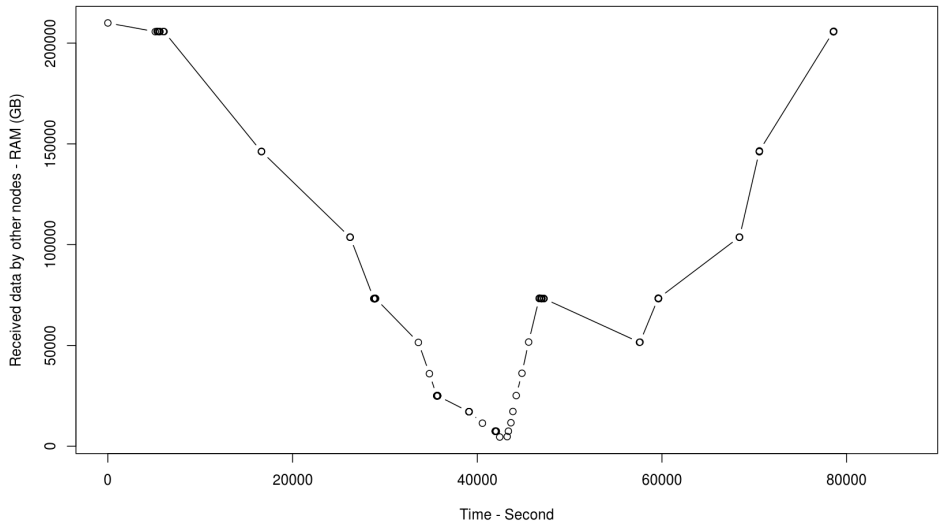

Figure F.74: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.005 and a growth factor f = 1.4.

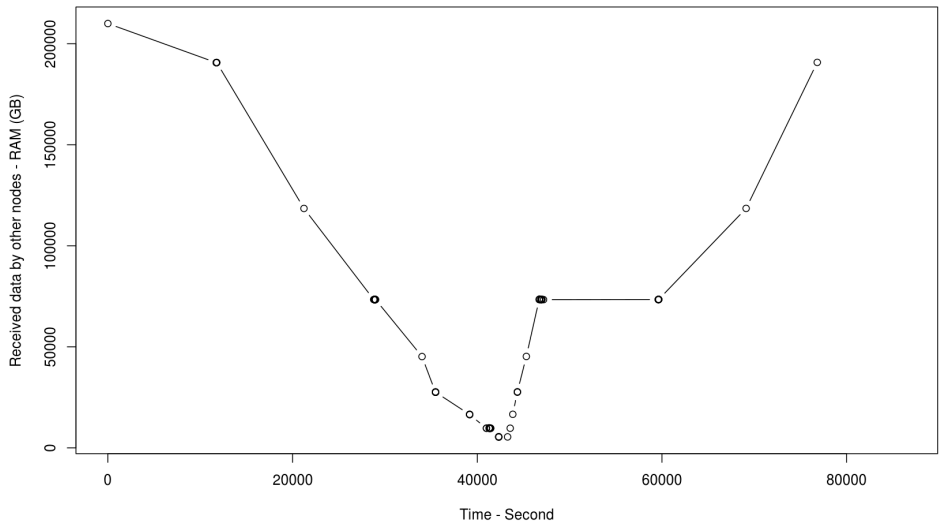

Figure F.75: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.005 and a growth factor f = 1.6.

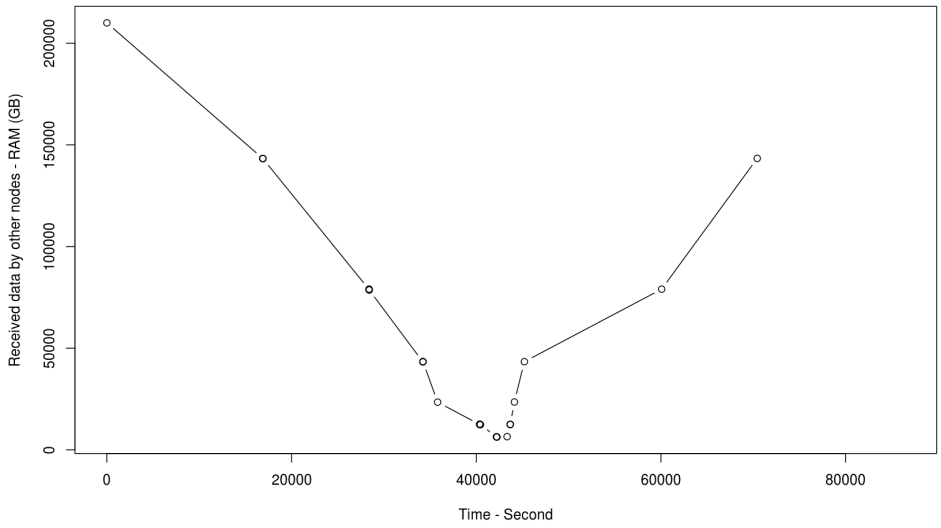

Figure F.76: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.005 and a growth factor f = 1.8.

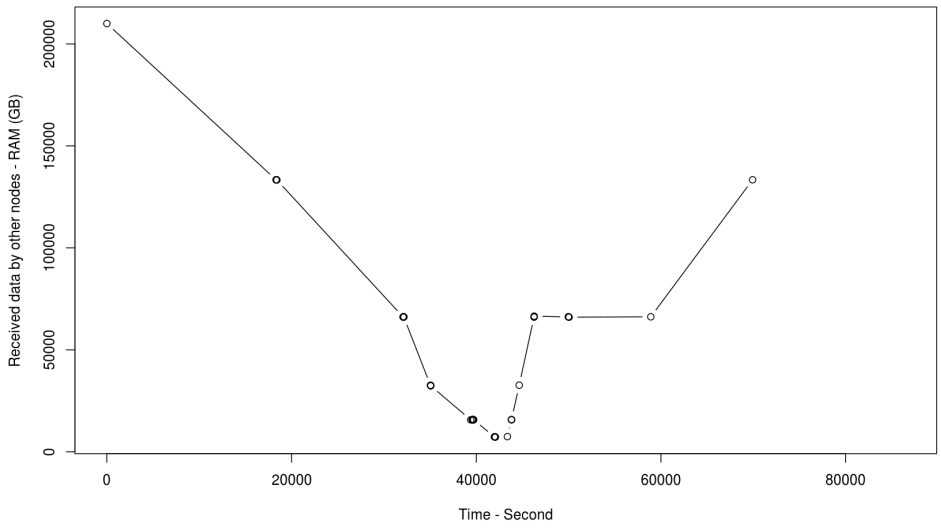

Figure F.77: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.005 and a growth factor f = 2.

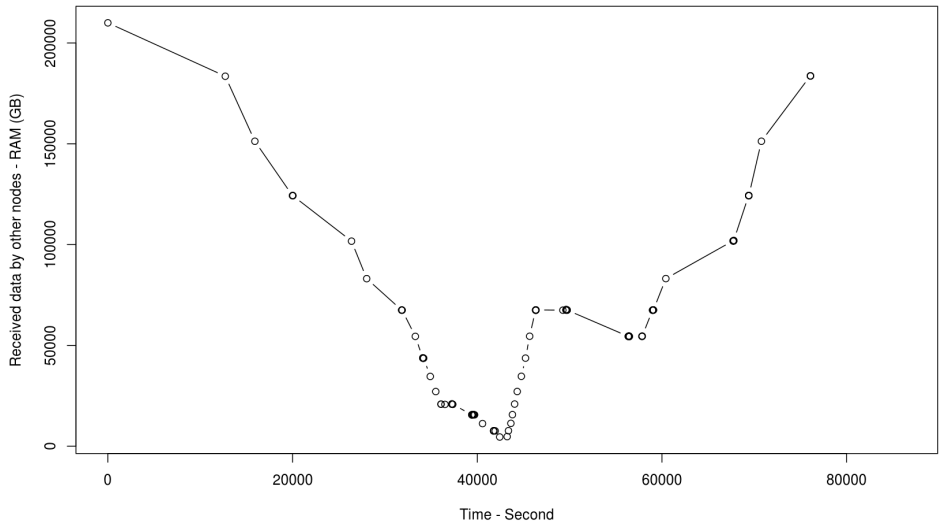

Figure F.78: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.01 and a growth factor f = 1.2.

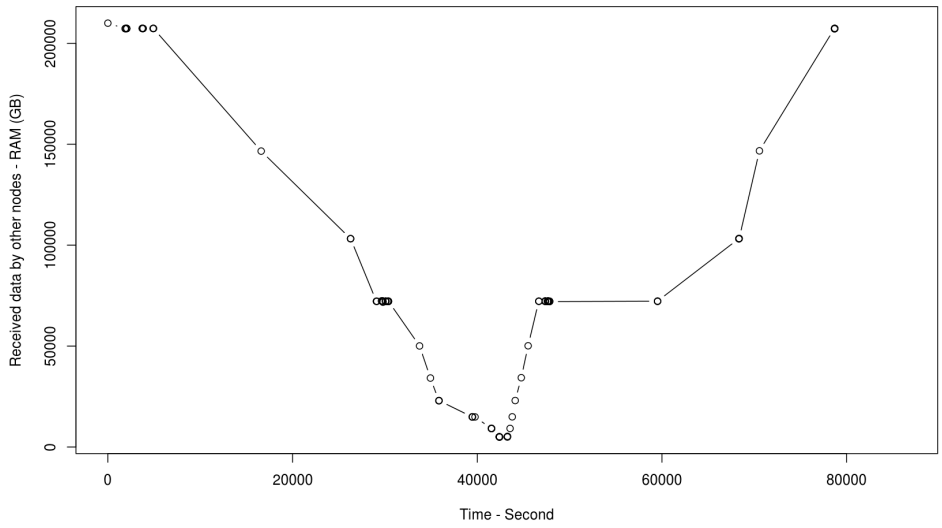

Figure F.79: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.01 and a growth factor f = 1.4.

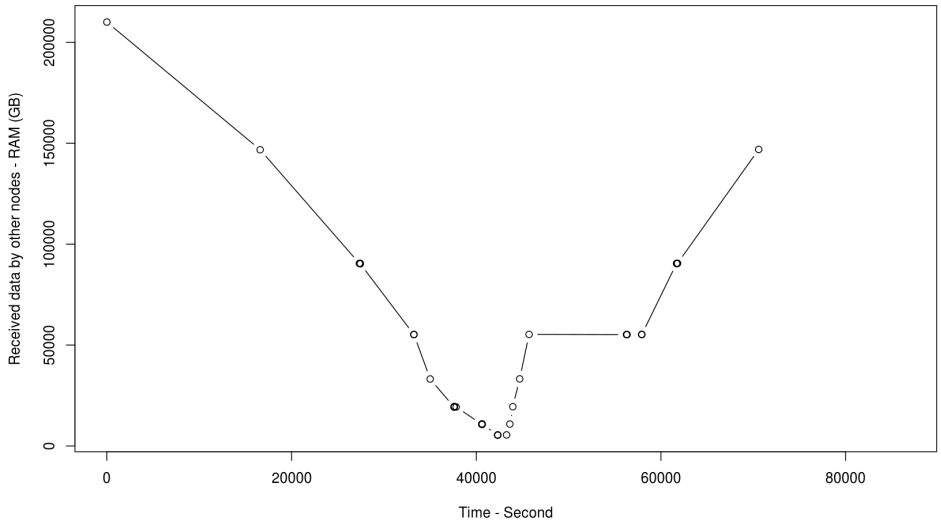

Figure F.80: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.01 and a growth factor f = 1.6.

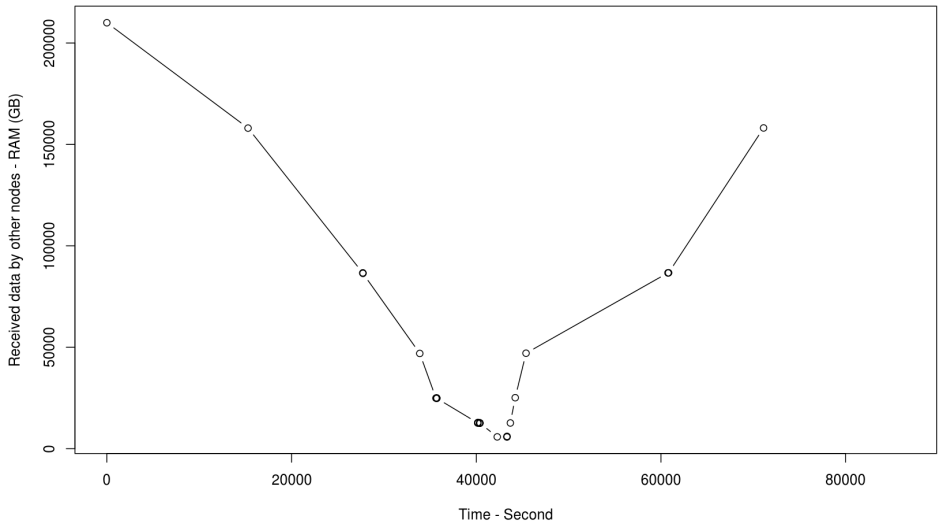

Figure F.81: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.01 and a growth factor f = 1.8.

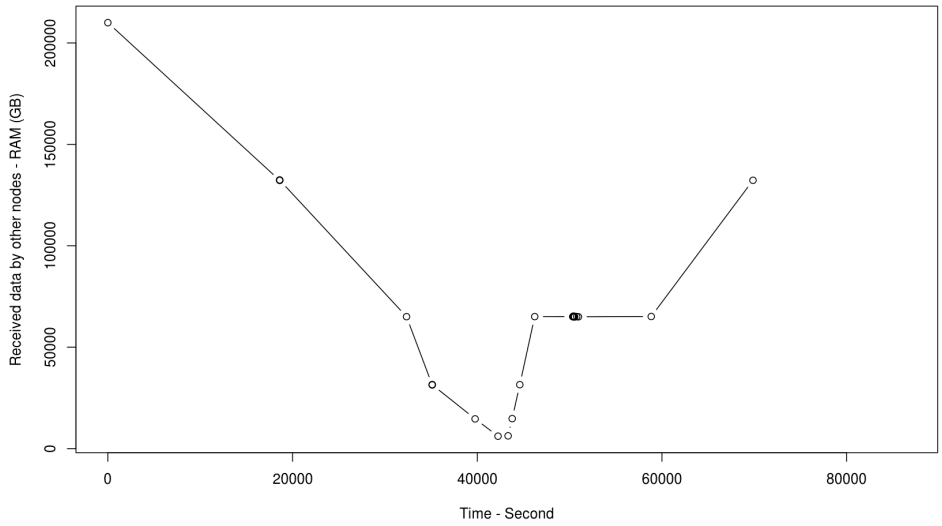

Figure F.82: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.01 and a growth factor f = 2.

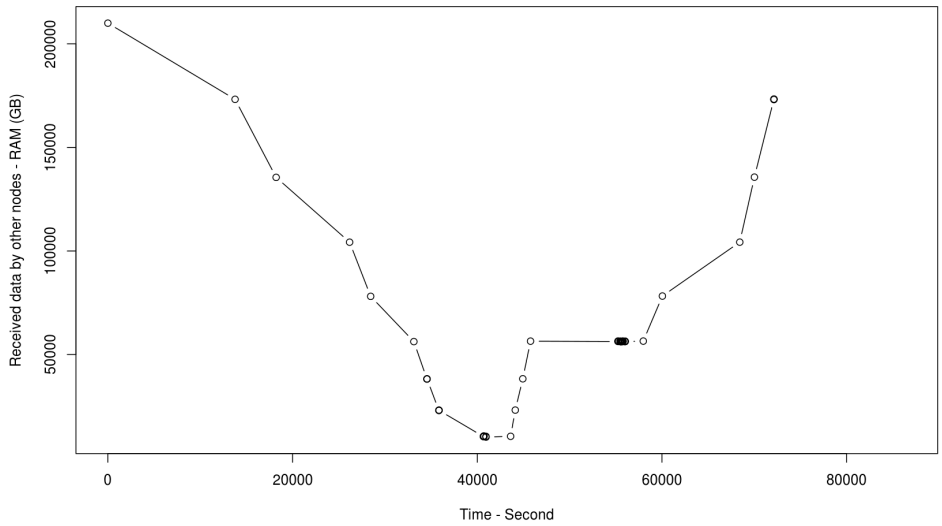

Figure F.83: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.05 and a growth factor f = 1.2.

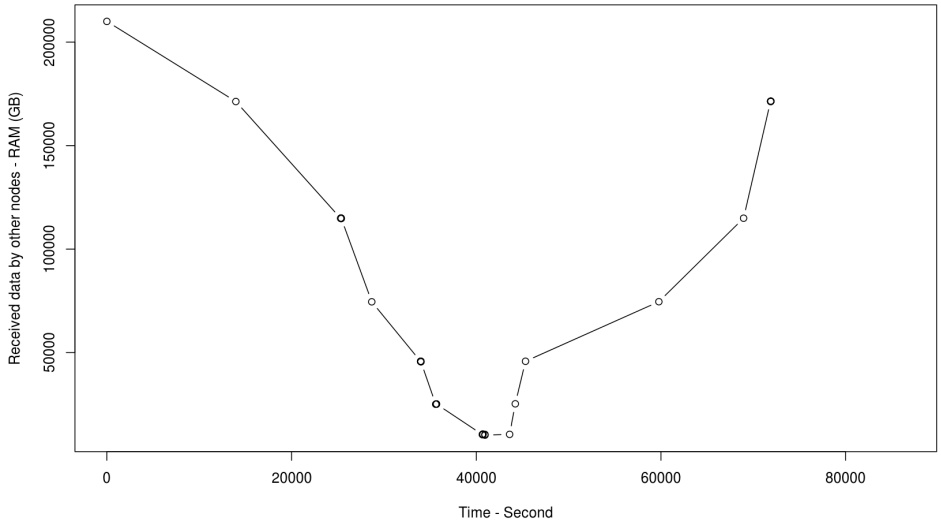

Figure F.84: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.05 and a growth factor f = 1.4.

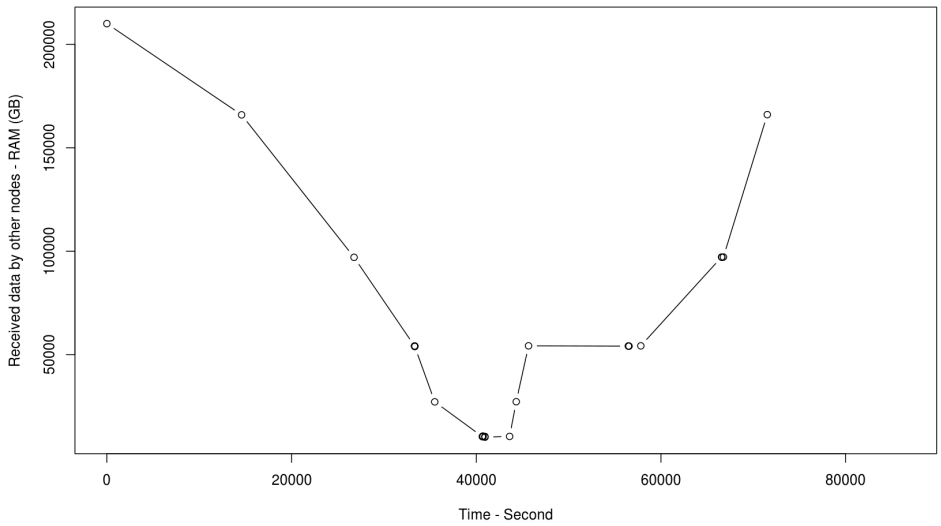

Figure F.85: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.05 and a growth factor f = 1.6.

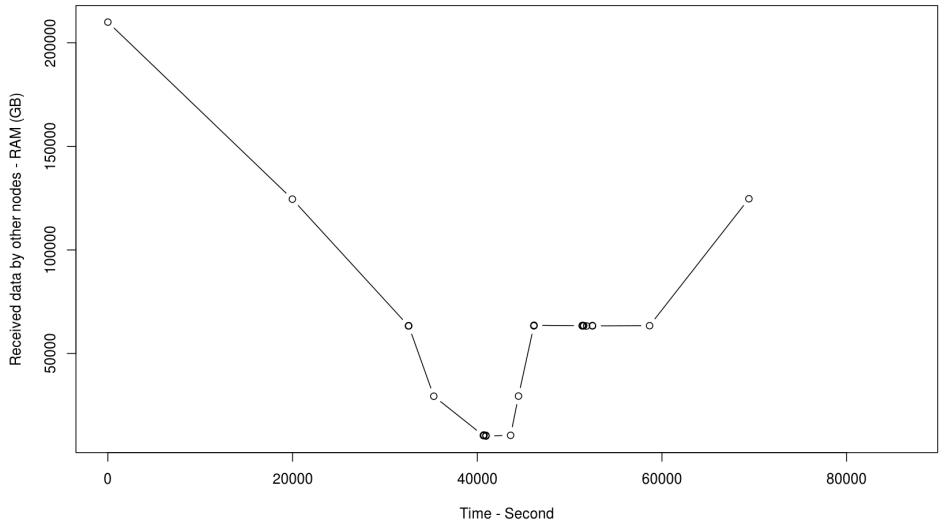

Figure F.86: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor $\beta = 0.05$ and a growth factor $f = 1.8$.

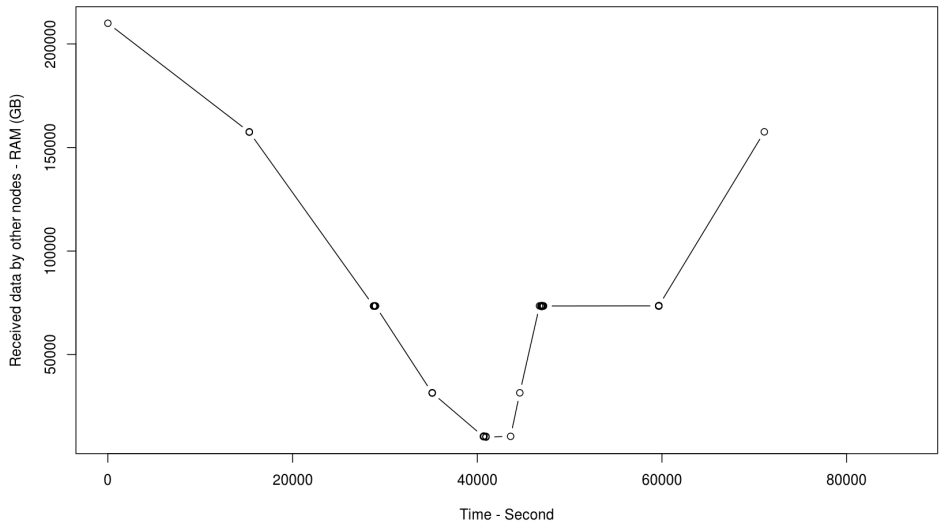

Figure F.87: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor $\beta = 0.05$ and a growth factor $f = 2$.

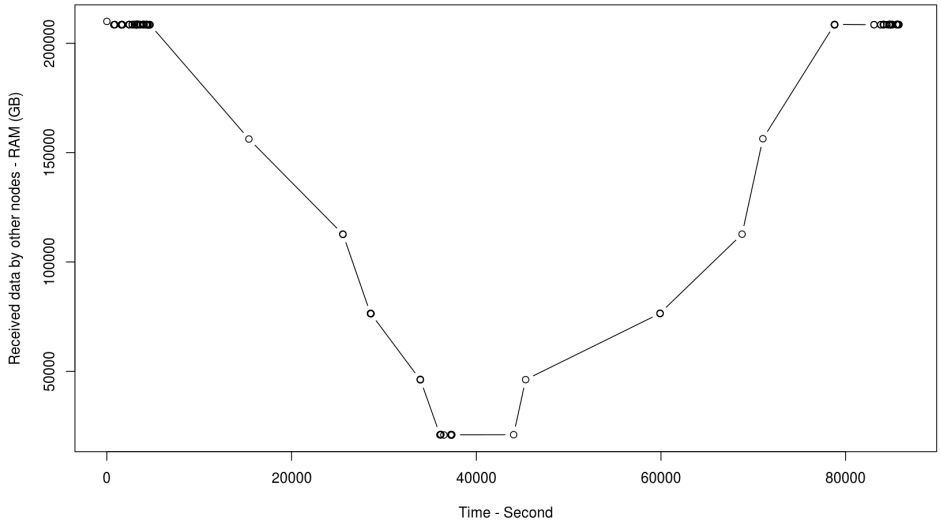

Figure F.88: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.1 and a growth factor f = 1.2.

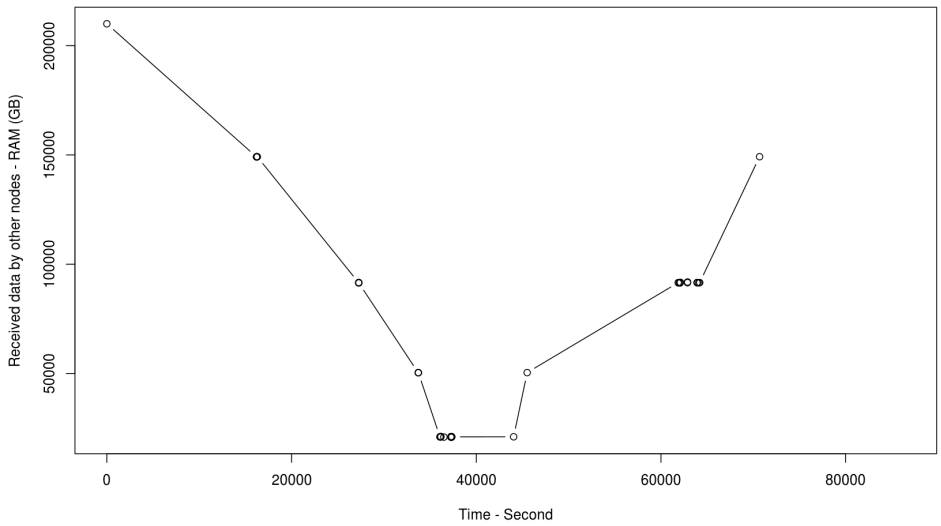

Figure F.89: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.1 and a growth factor f = 1.4.

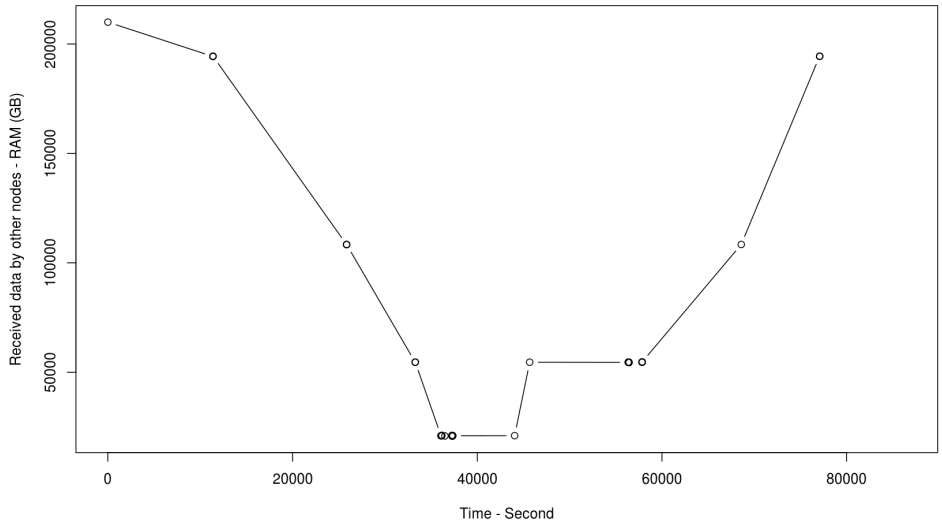

Figure F.90: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.1 and a growth factor f = 1.6.

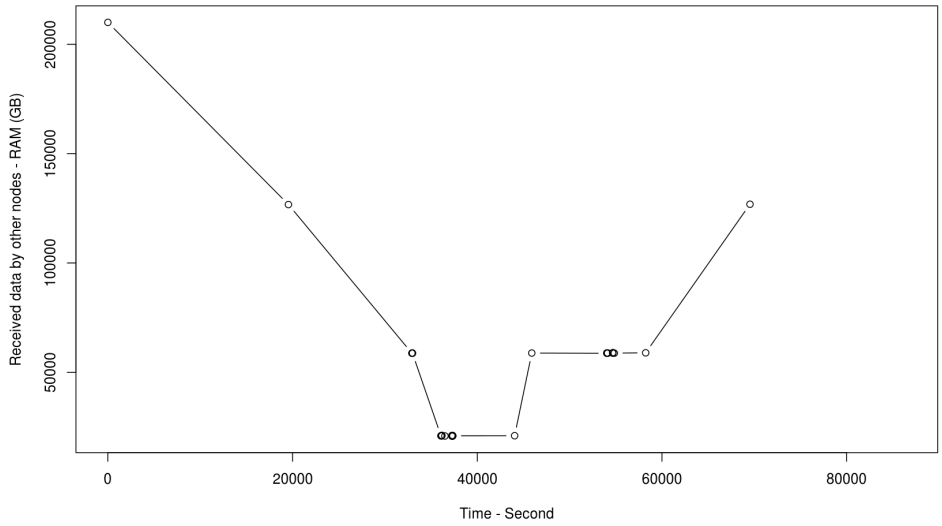

Figure F.91: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor β = 0.1 and a growth factor f = 1.8.

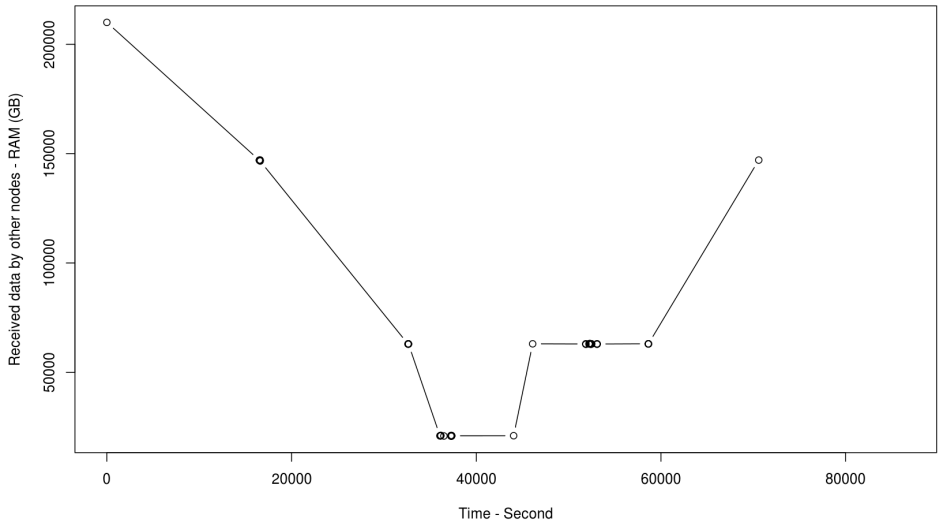

Figure F.92: Exponential-sized class-based update policy. How a cloud management system views data center's *RAM* capacity when a base factor $\beta = 0.1$ and a growth factor $f = 2$.

F.3 Data center's storage

Table F.3: Exponential-sized class-based update policy. Number of Cloud LSA updates based on changes in a proposed data center's **storage**. In this Table "β" refers to a *"base factor"*, "*f*" refers to *growth factor*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

β	*f*	**Number of updates**					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.00001	1.2	49	1	1	15	127	23.448
0.00001	1.4	23	1	1	7	93	15.741
0.00001	1.6	16	2	2	5	97	13.139
0.00001	1.8	12	2	2	5	47	8.530
0.00001	2	12	2	2	3	53	12.711
0.00005	1.2	49	1	1	15	155	26.081
0.00005	1.4	28	1	1	5	93	16.797
0.00005	1.6	18	2	2	3	61	14.206
0.00005	1.8	14	2	2	3	53	10.59
0.00005	2	13	2	2	3	49	9.488
0.0001	1.2	48	1	1	13	127	20.798
0.0001	1.4	29	1	1	5	91	17.804
0.0001	1.6	12	2	2	5	41	6.480
0.0001	1.8	11	2	2	5	45	7.211
0.0001	2	13	2	2	3	49	9.364
0.0005	1.2	45	1	1	11	123	22.086

Continued on next page

Table F.3 – continued from previous page

β	f	Number of updates					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.0005	1.4	32	1	1	5	87	19.148
0.0005	1.6	26	2	2	6	67	15.745
0.0005	1.8	20	2	2	3	121	18.885
0.0005	2	9	2	2	3	41	7.952
0.001	1.2	82	1	1	17	203	37.216
0.001	1.4	26	1	1	5	89	16.044
0.001	1.6	16	2	2	5	67	12.401
0.001	1.8	13	2	2	5	53	10.569
0.001	2	9	2	2	3	45	8.505
0.005	1.2	49	1	1	17	115	21.669
0.005	1.4	51	1	1	9	129	28.323
0.005	1.6	25	2	2	5	67	14.171
0.005	1.8	13	2	2	3	65	11.375
0.005	2	10	2	2	3	99	13.520
0.01	1.2	44	1	1	13	135	22.324
0.01	1.4	62	1	1	7	170	32.904
0.01	1.6	19	2	2	3	83	15.129
0.01	1.8	16	2	2	5	75	10.771
0.01	2	10	2	2	3	61	11.767
0.05	1.2	27	1	1	7	95	16.857
0.05	1.4	23	1	1	5	85	15.298
0.05	1.6	12	2	2	5	51	8.337
0.05	1.8	12	2	2	3	53	11.299

Continued on next page

Table F.3 – continued from previous page

β	f	Number of updates					
		Mean	**UB**	**LB**	**Min**	**Max**	**Std.**
0.05	2	13	2	2	3	73	11.990
0.1	1.2	102	1	1	24	247	45.553
0.1	1.4	18	1	1	3	101	15.106
0.1	1.6	19	2	2	5	107	14.147
0.1	1.8	12	2	2	3	73	12.901
0.1	2	12	2	2	3	59	11.882

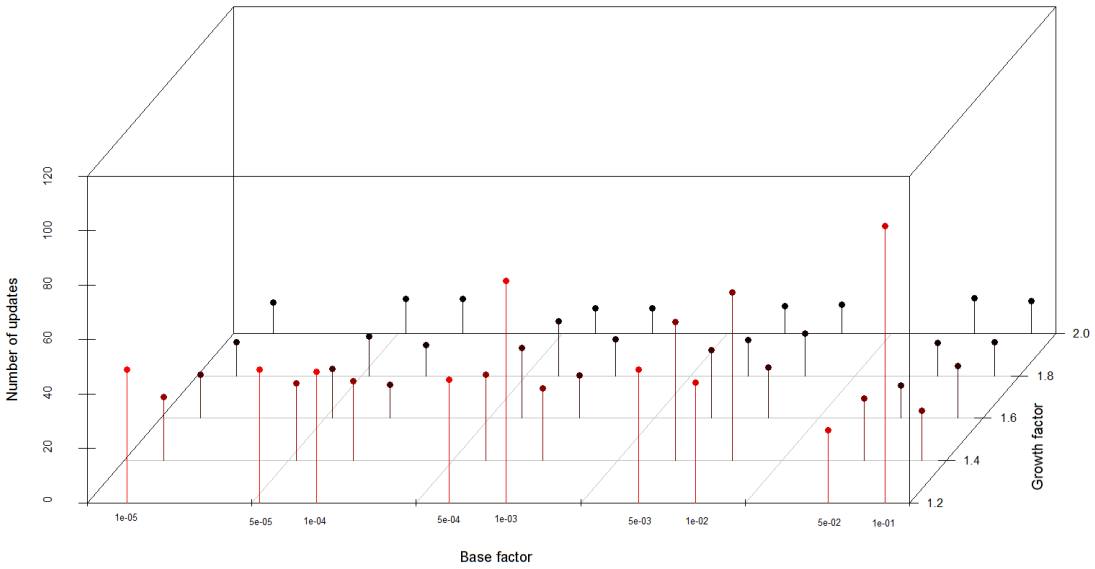

Figure F.93: Exponential-sized class-based update policy. Number of updates per different growth factor and base factor values based on changes of a data center's *storage* capacity. The "X" axis is in logarithm scale.

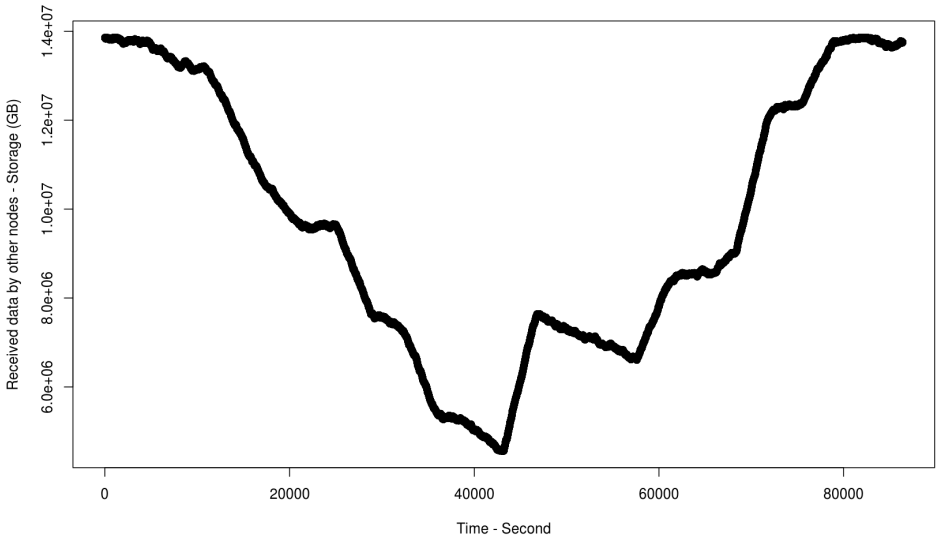

Figure F.94: Data center sample *storage* (GB) capacity per second

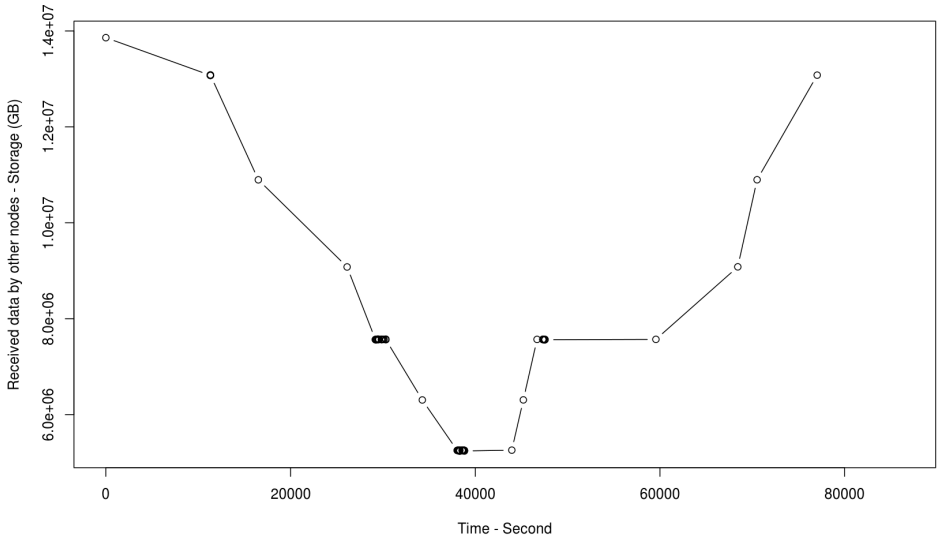

Figure F.95: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-05 and a growth factor f = 1.2.

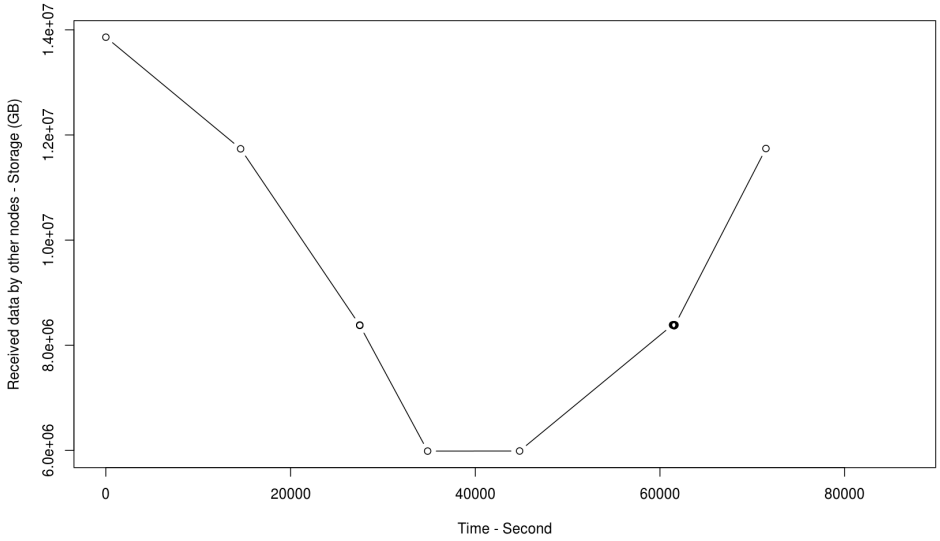

Figure F.96: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-05 and a growth factor f = 1.4.

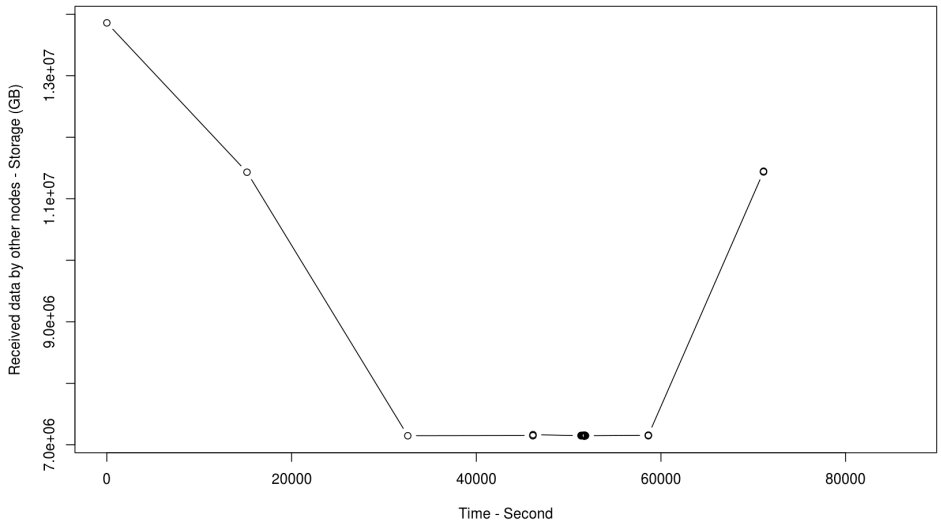

Figure F.97: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-05 and a growth factor f = 1.6.

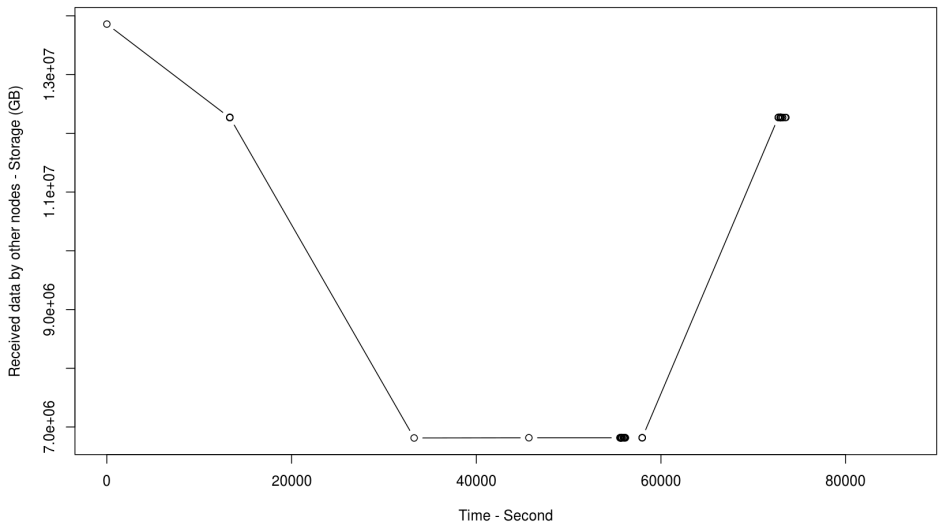

Figure F.98: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-05 and a growth factor f = 1.8.

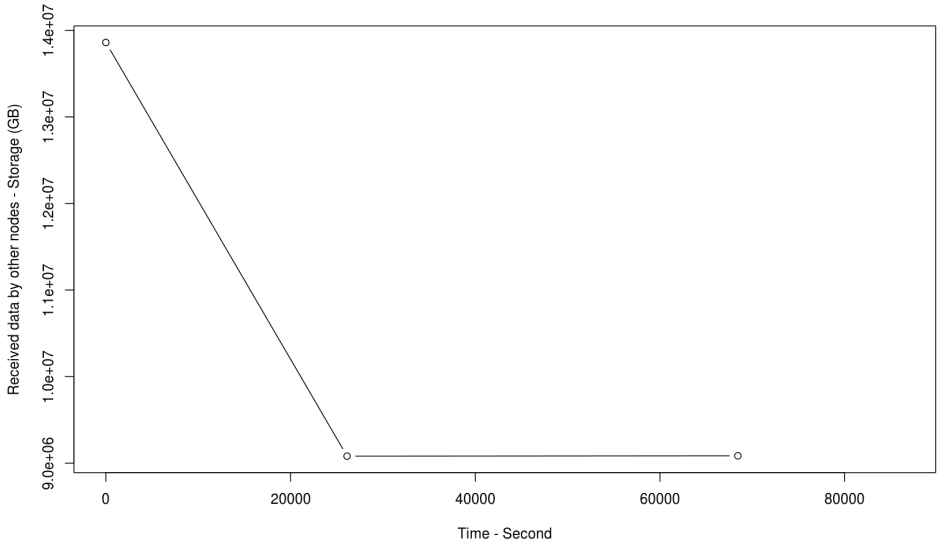

Figure F.99: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-05 and a growth factor f = 2.

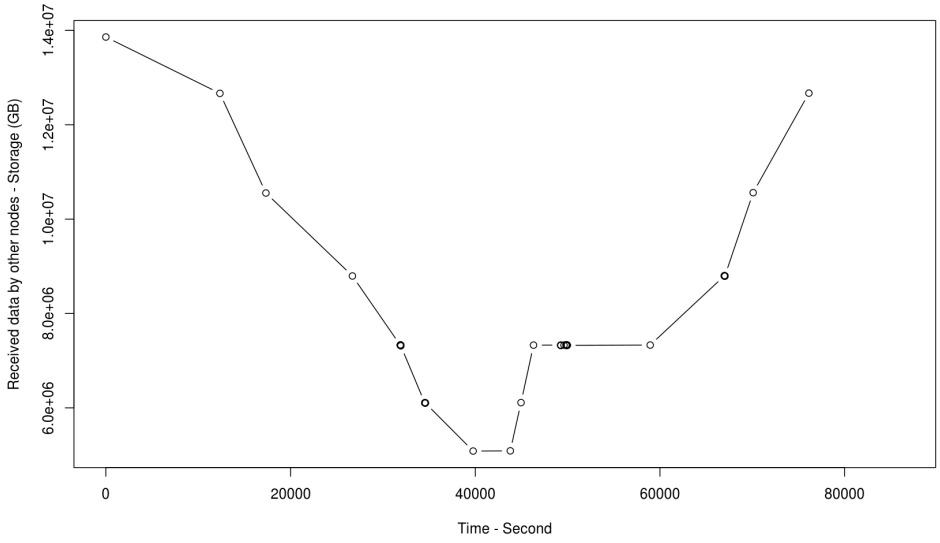

Figure F.100: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-05 and a growth factor f = 1.2.

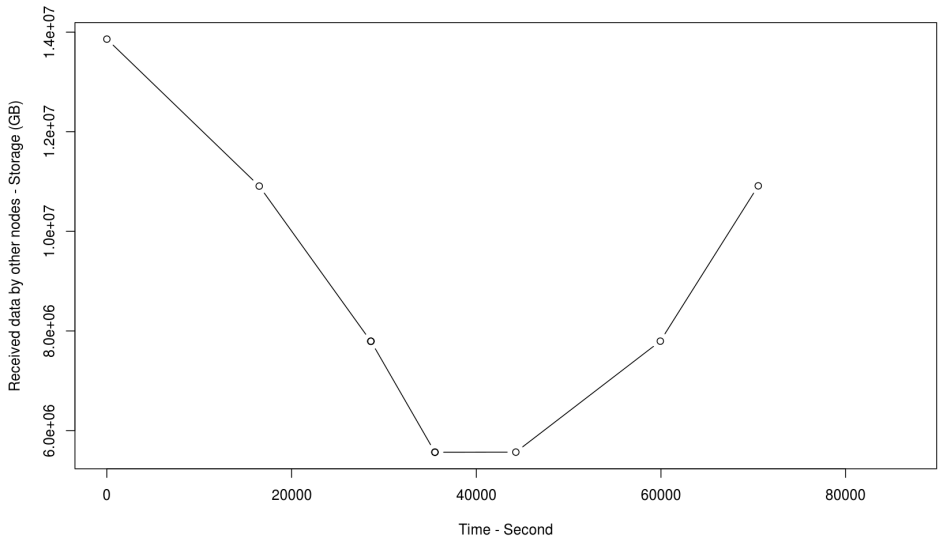

Figure F.101: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-05 and a growth factor f = 1.4.

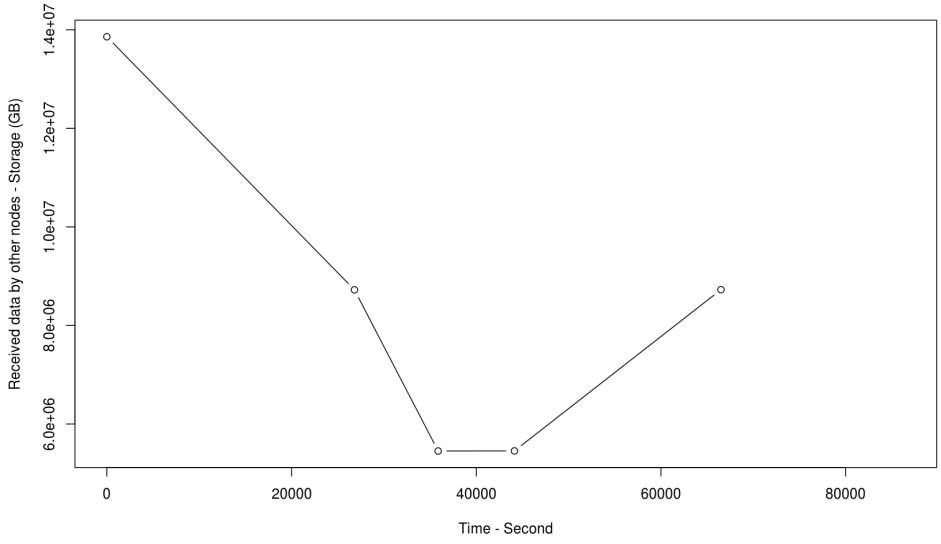

Figure F.102: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-05 and a growth factor f = 1.6.

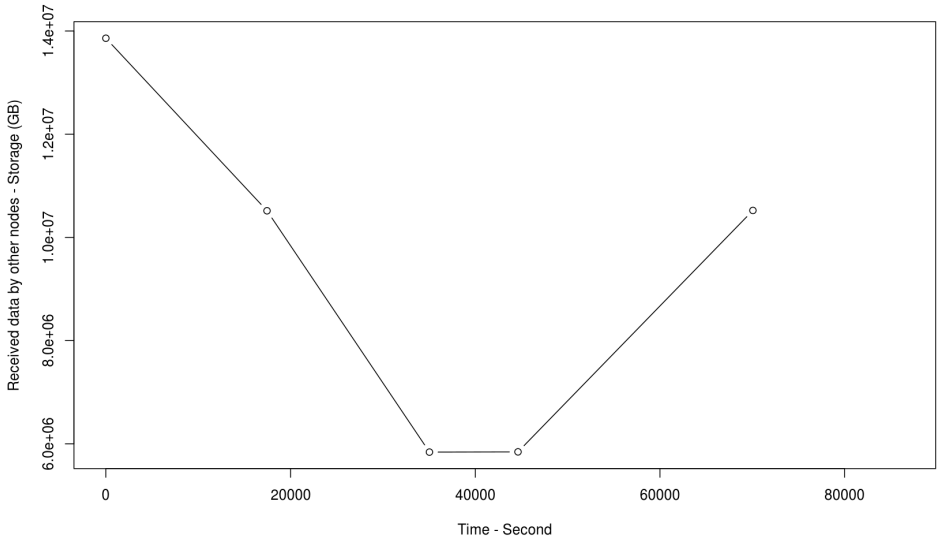

Figure F.103: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-05 and a growth factor f = 1.8.

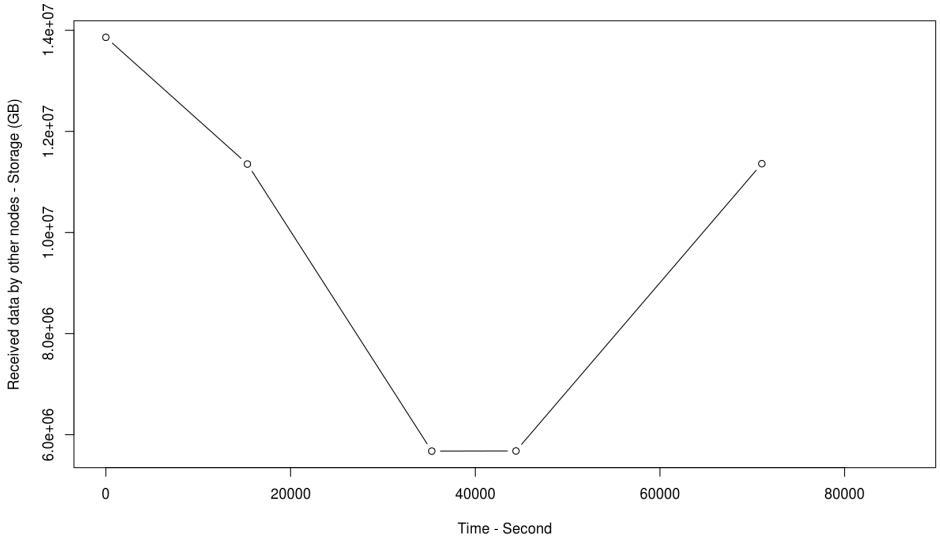

Figure F.104: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-05 and a growth factor f = 2.

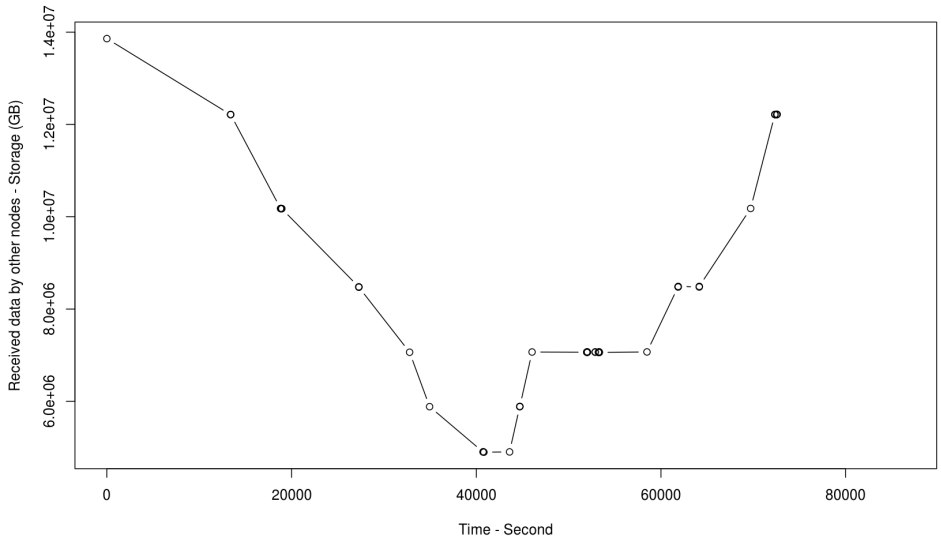

Figure F.105: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-04 and a growth factor f = 1.2.

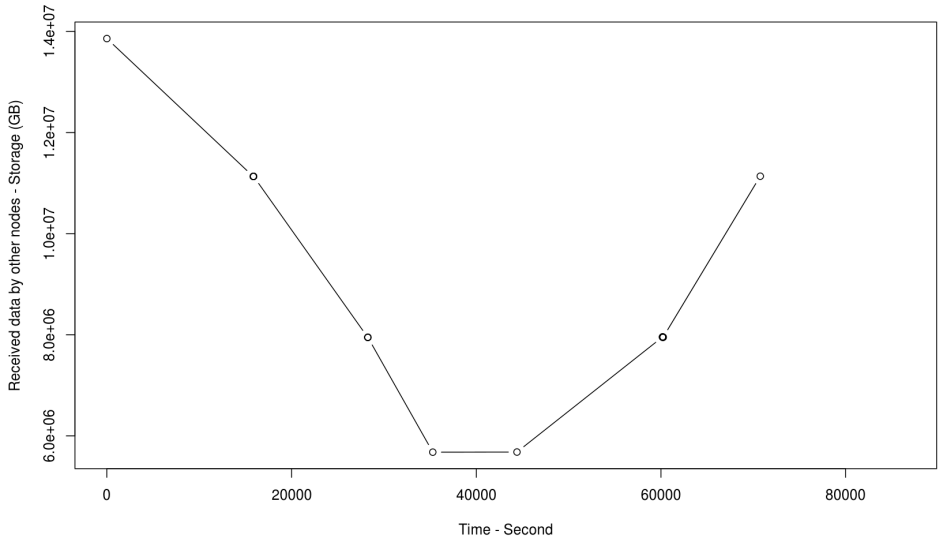

Figure F.106: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-04 and a growth factor f = 1.4.

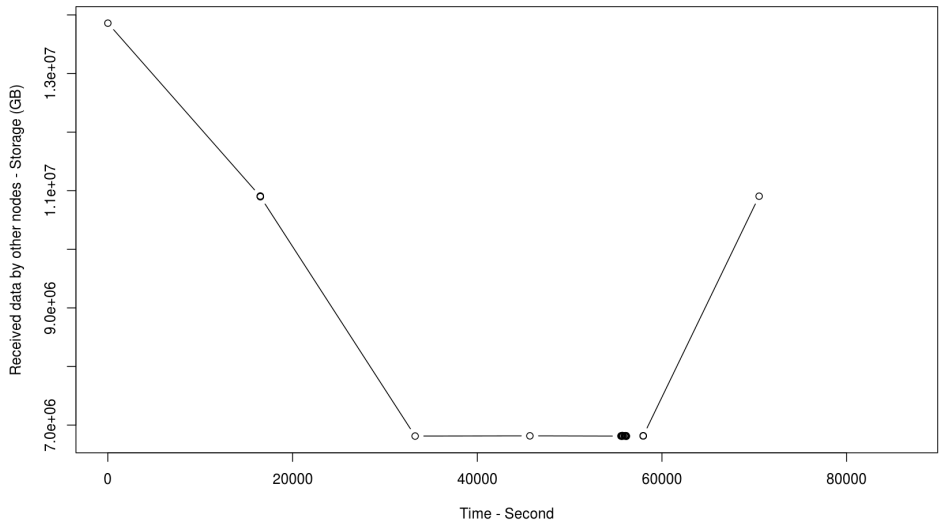

Figure F.107: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-04 and a growth factor f = 1.6.

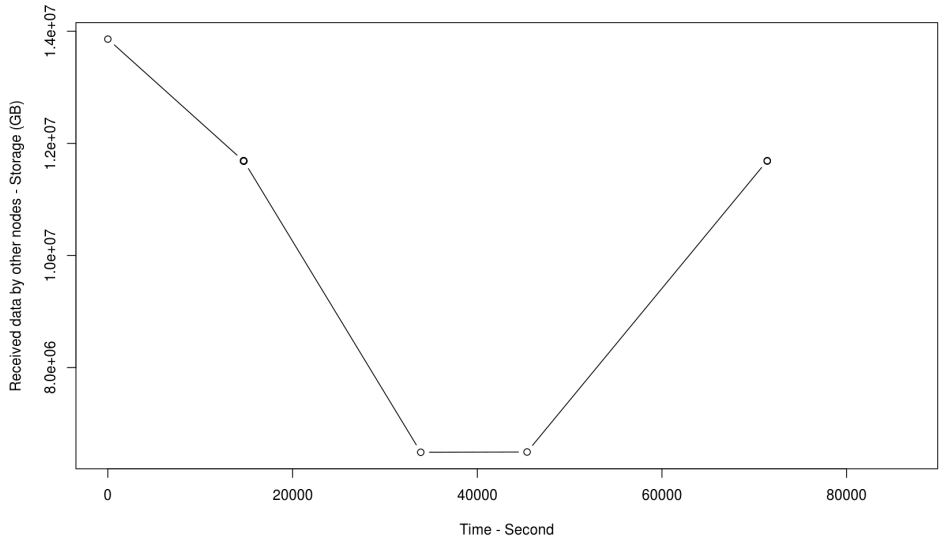

Figure F.108: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-04 and a growth factor f = 1.8.

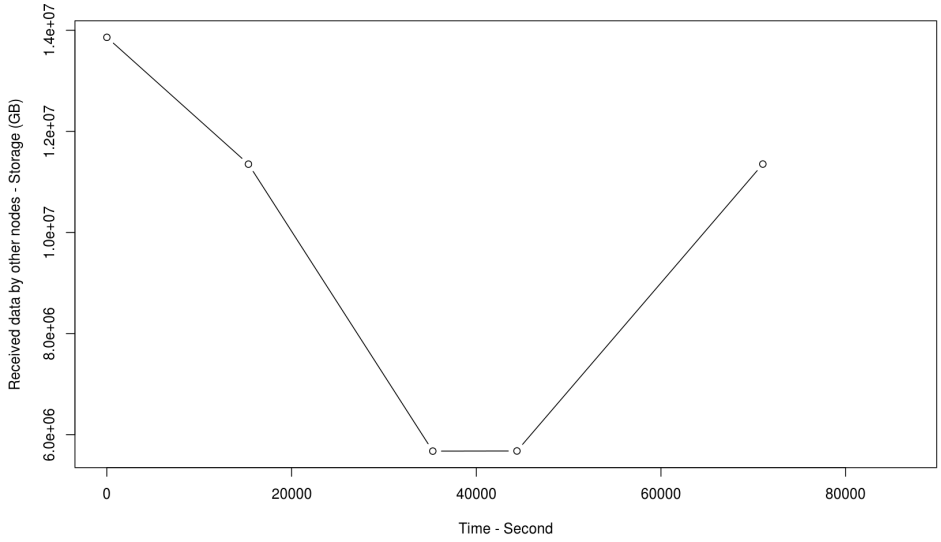

Figure F.109: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 1e-04 and a growth factor f = 2.

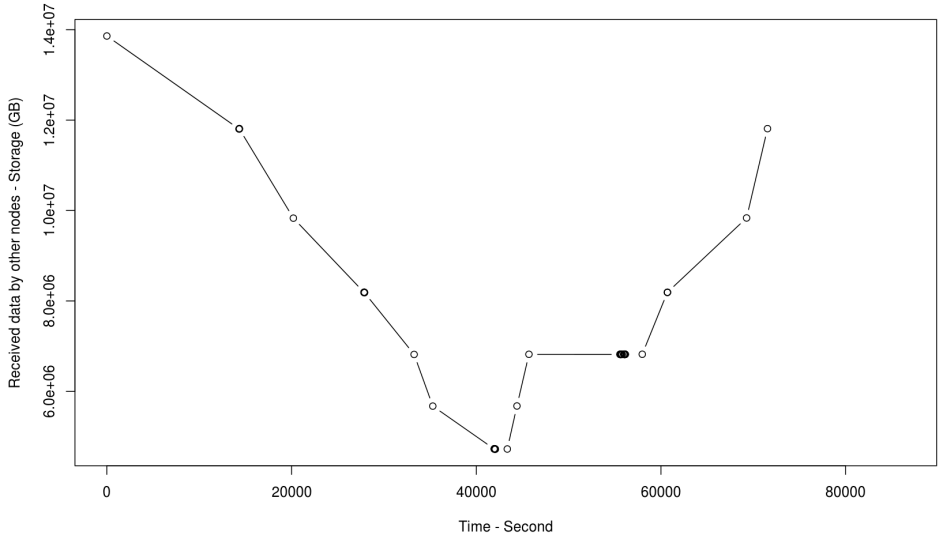

Figure F.110: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-04 and a growth factor f = 1.2.

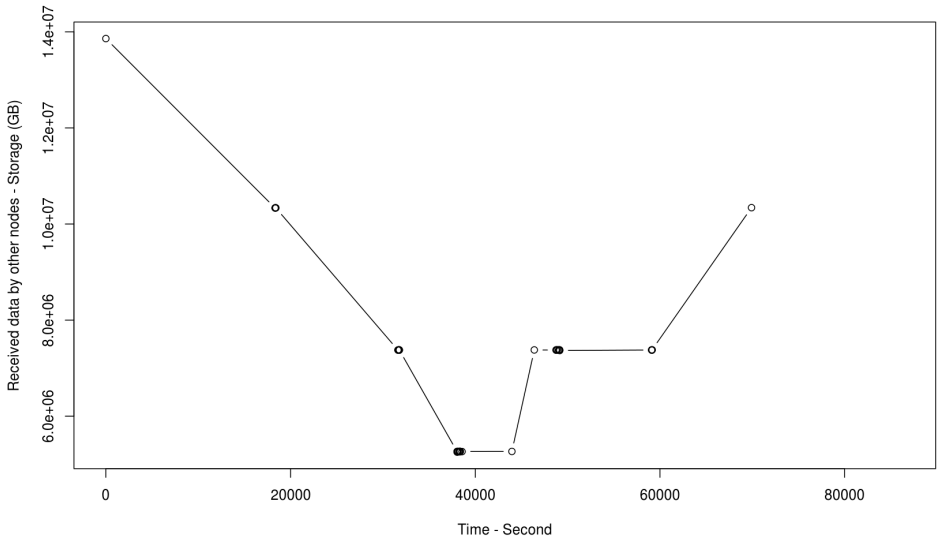

Figure F.111: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-04 and a growth factor f = 1.4.

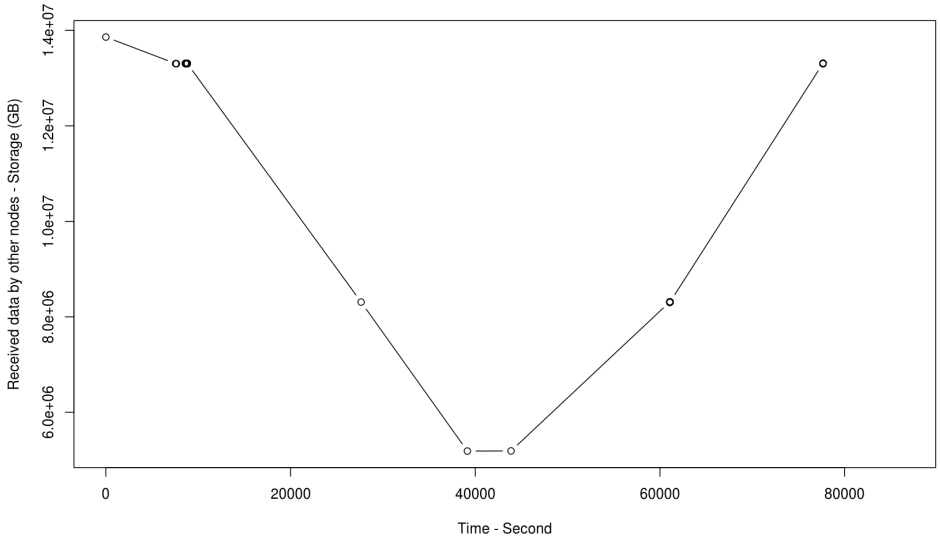

Figure F.112: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-04 and a growth factor f = 1.6.

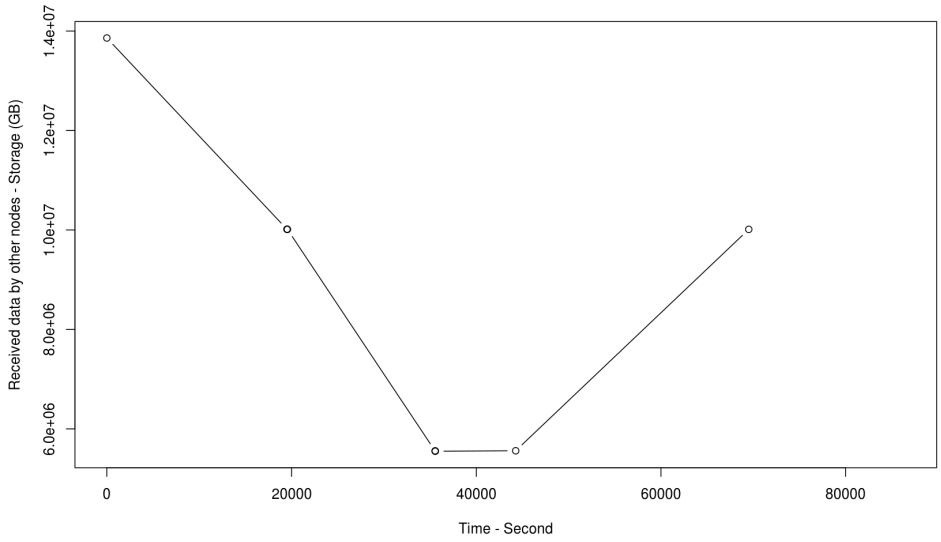

Figure F.113: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-04 and a growth factor f = 1.8.

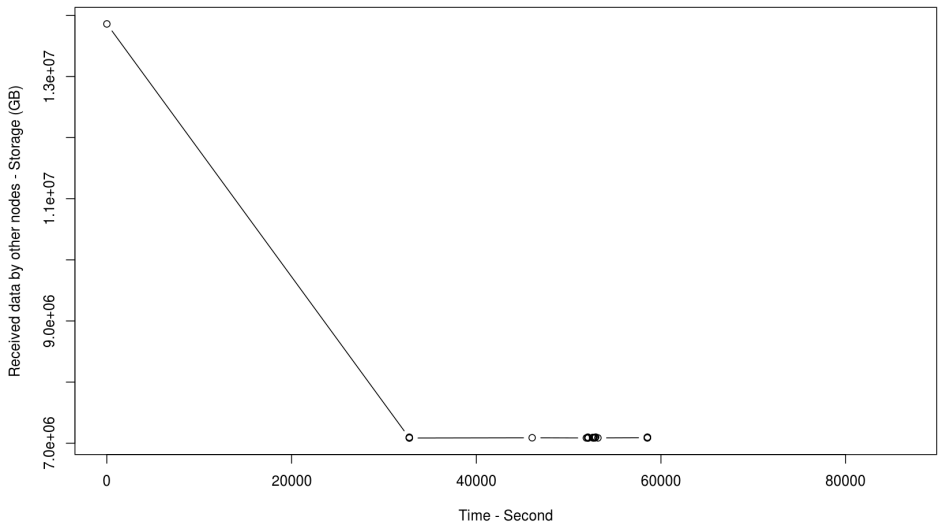

Figure F.114: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 5e-04 and a growth factor f = 2.

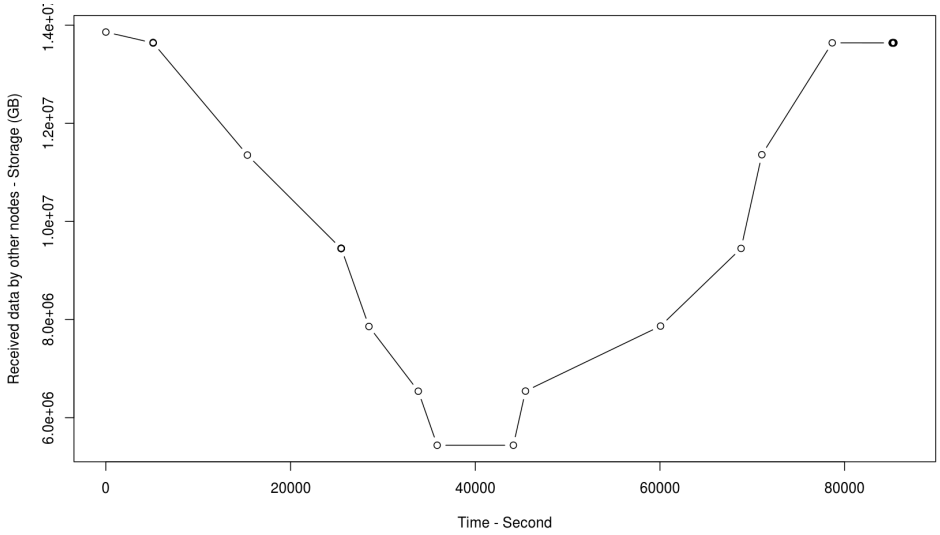

Figure F.115: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.001 and a growth factor f = 1.2.

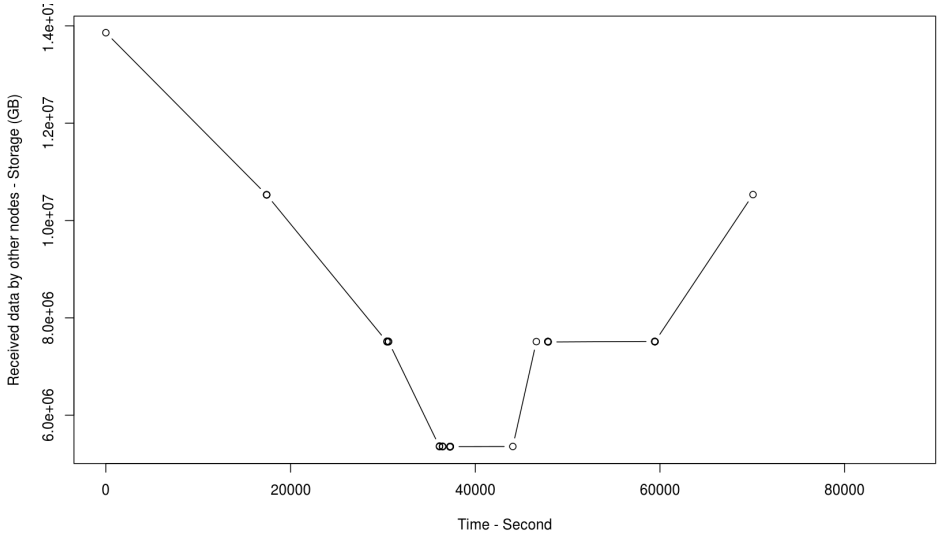

Figure F.116: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.001 and a growth factor f = 1.4.

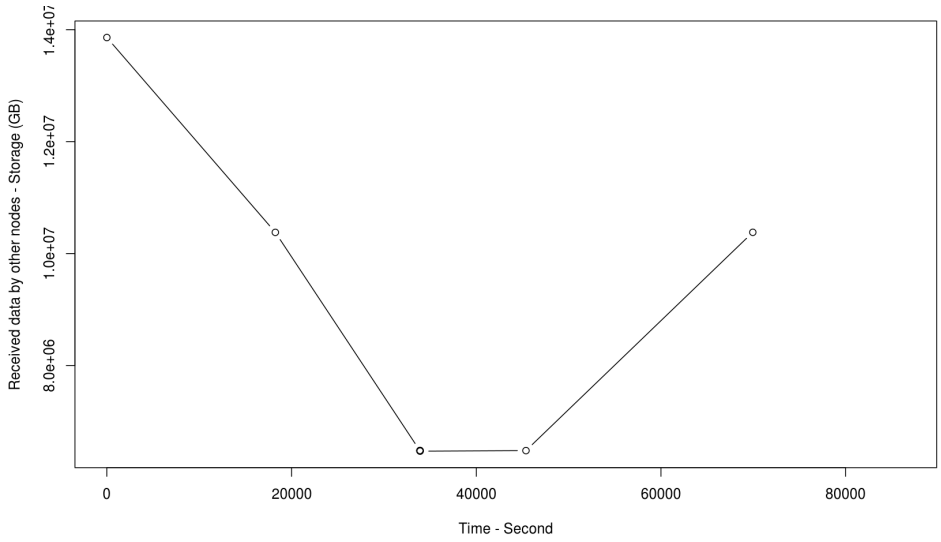

Figure F.117: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.001 and a growth factor f = 1.6.

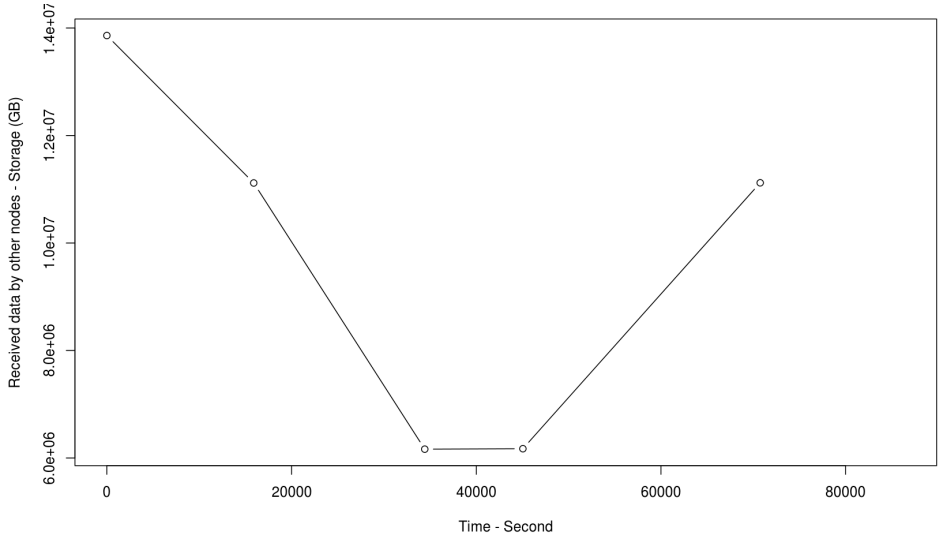

Figure F.118: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.001 and a growth factor f = 1.8.

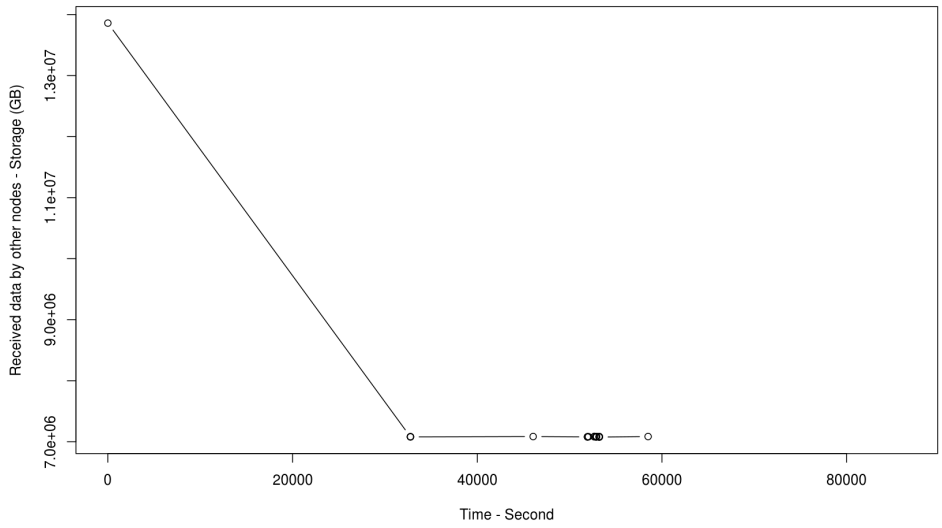

Figure F.119: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.001 and a growth factor f = 2.

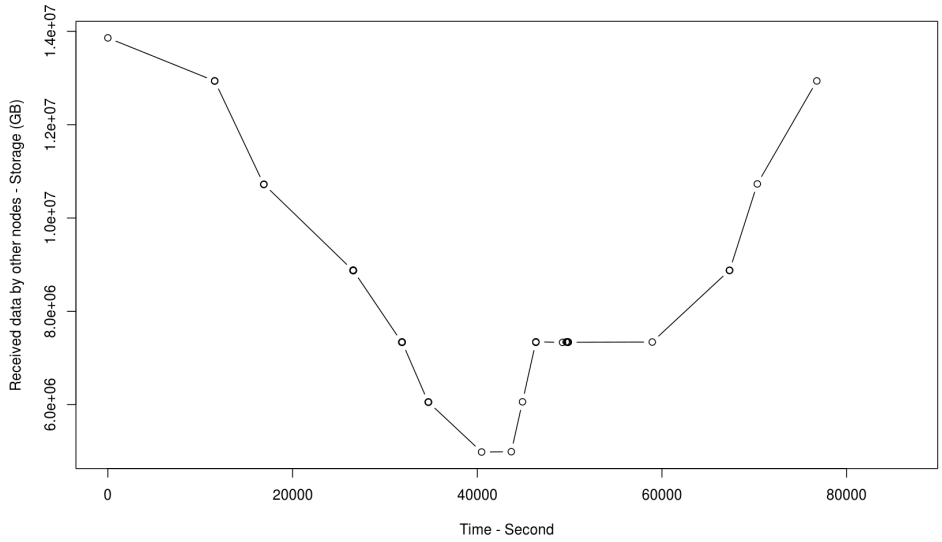

Figure F.120: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.005 and a growth factor f = 1.2.

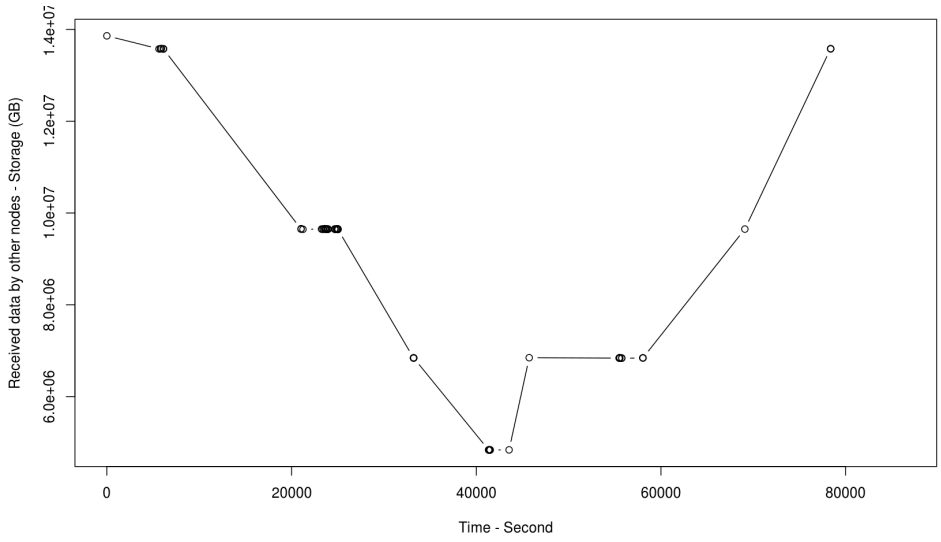

Figure F.121: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.005 and a growth factor f = 1.4.

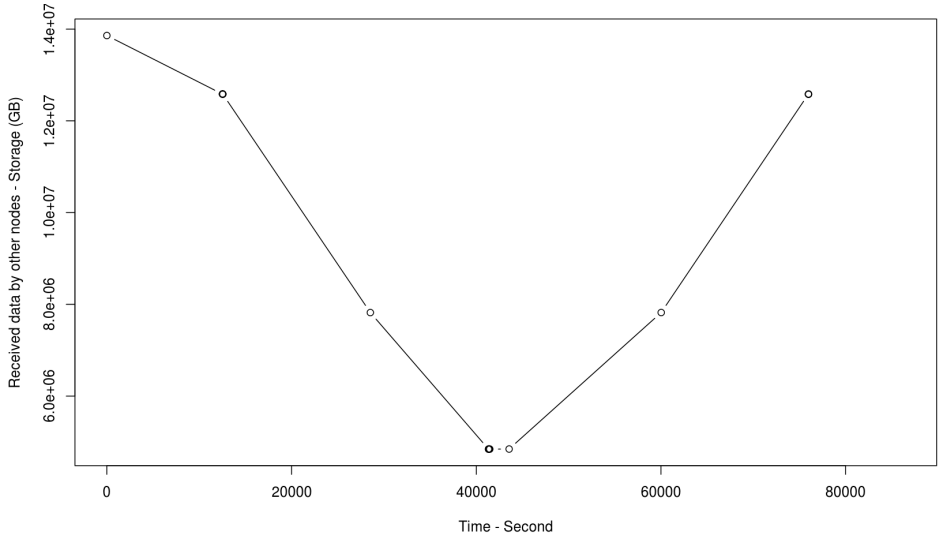

Figure F.122: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.005 and a growth factor f = 1.6.

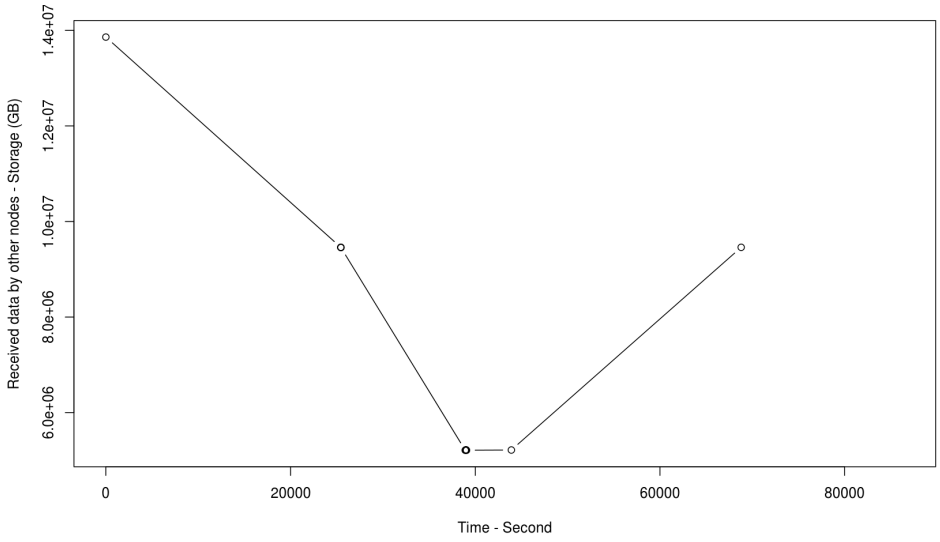

Figure F.123: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.005 and a growth factor f = 1.8.

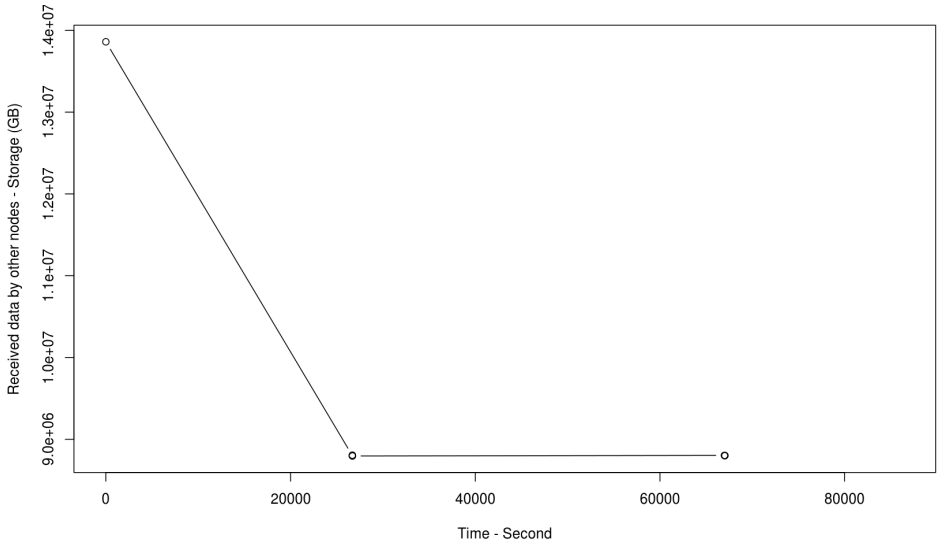

Figure F.124: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.005 and a growth factor f = 2.

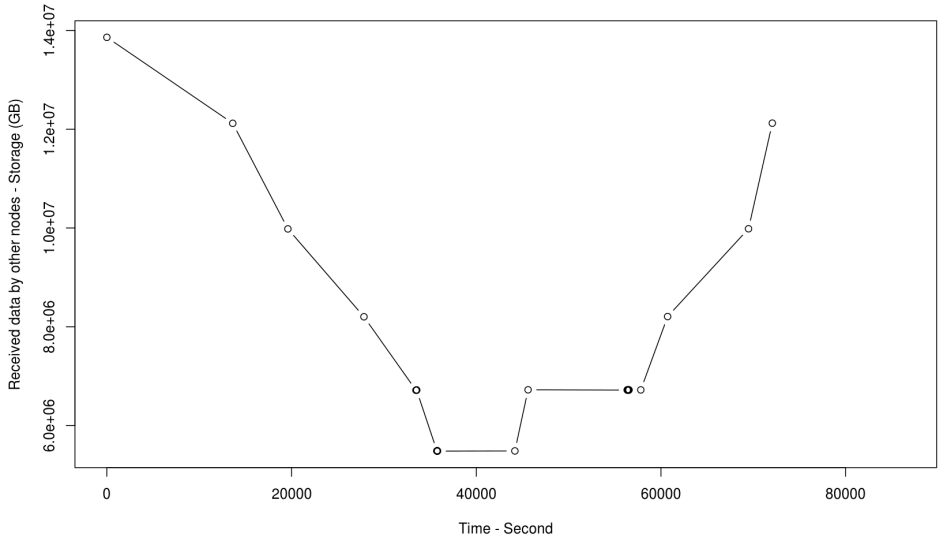

Figure F.125: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.01 and a growth factor f = 1.2.

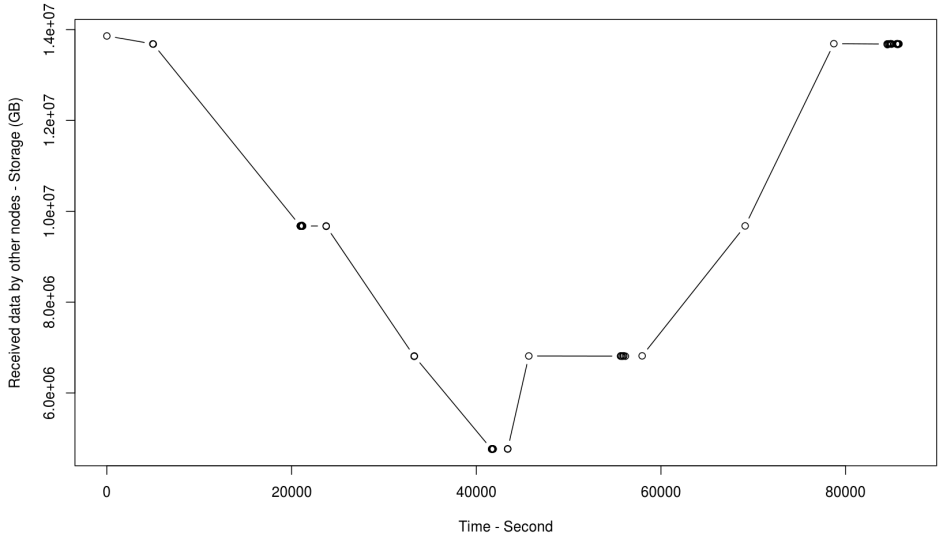

Figure F.126: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.01 and a growth factor f = 1.4.

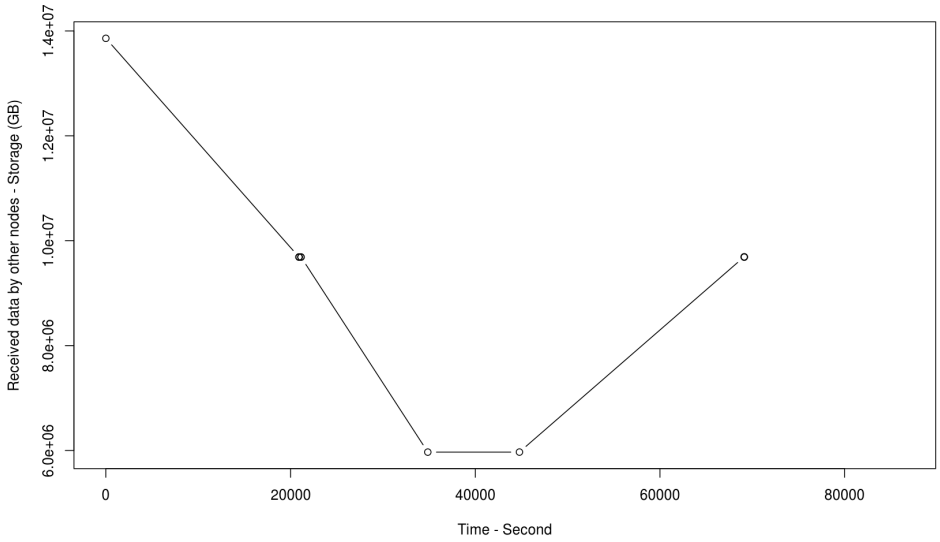

Figure F.127: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.01 and a growth factor f = 1.6.

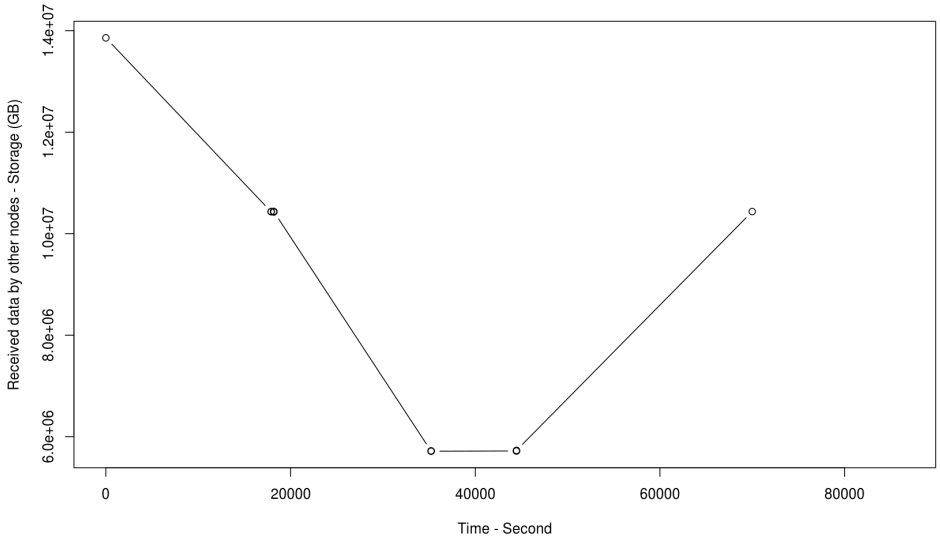

Figure F.128: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.01 and a growth factor f = 1.8.

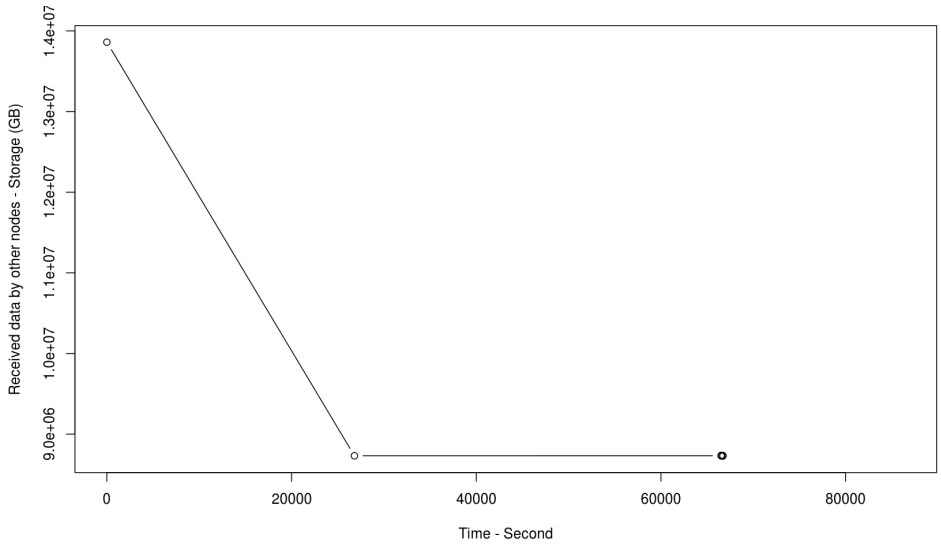

Figure F.129: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.01 and a growth factor $f = 2$.

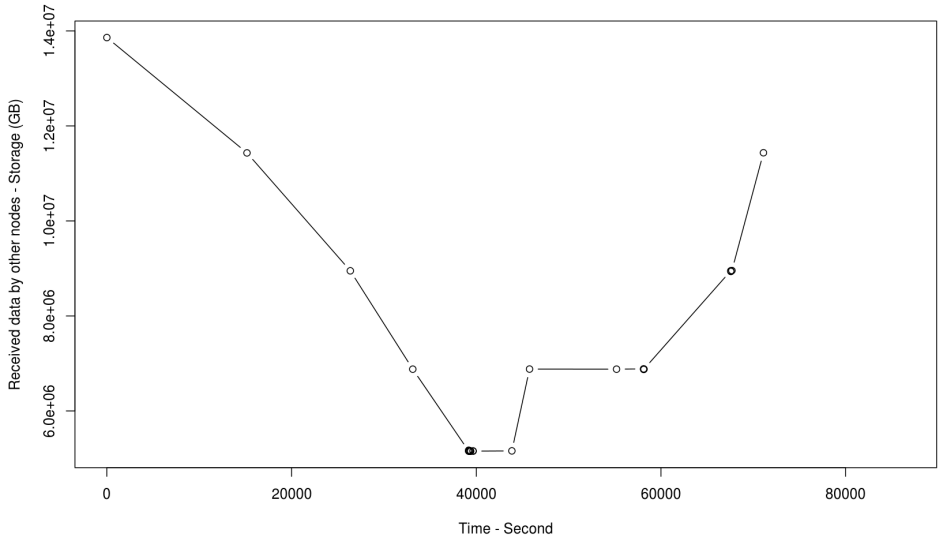

Figure F.130: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.05 and a growth factor $f = 1.2$.

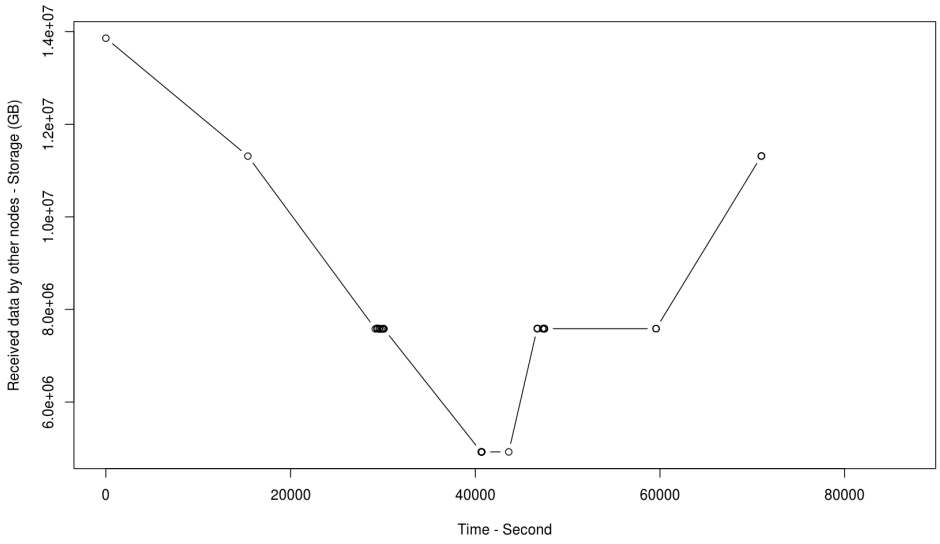

Figure F.131: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.05 and a growth factor f = 1.4.

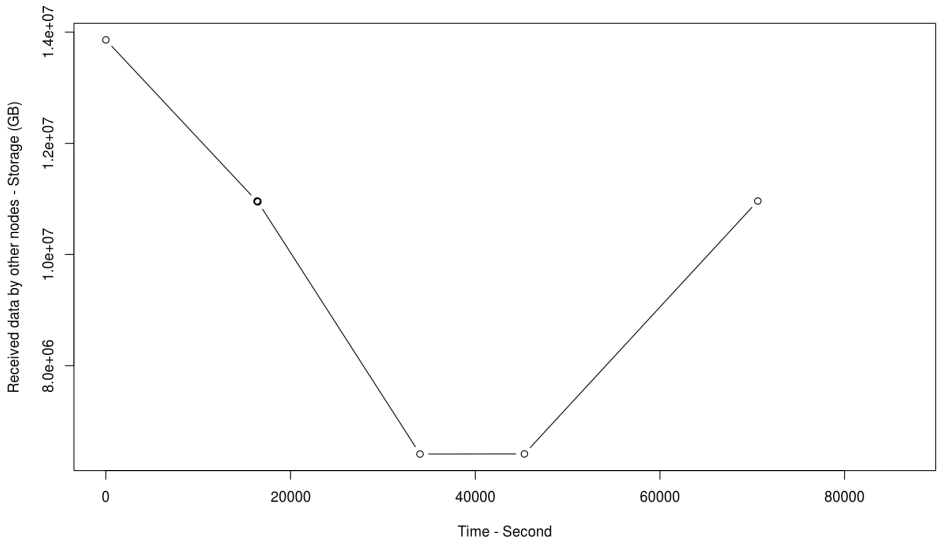

Figure F.132: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.05 and a growth factor f = 1.6.

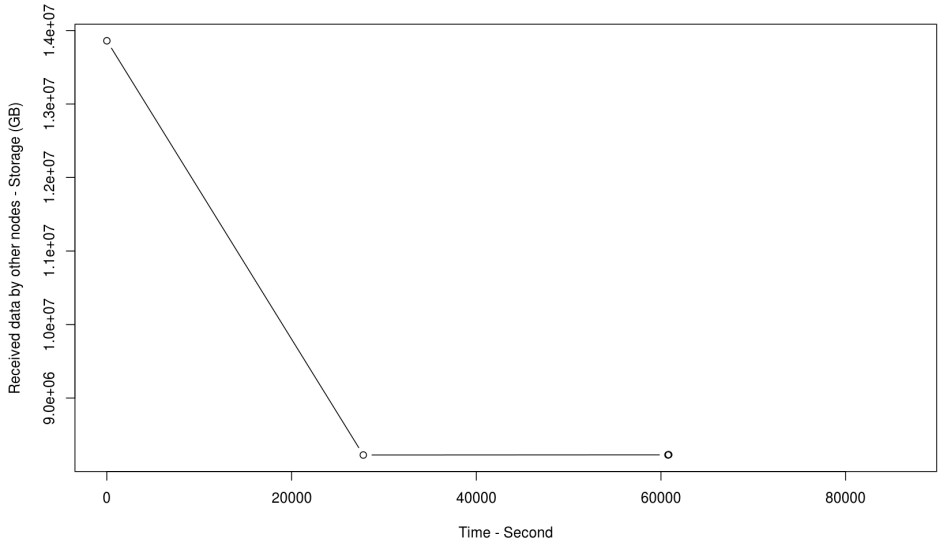

Figure F.133: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.05 and a growth factor $f = 1.8$.

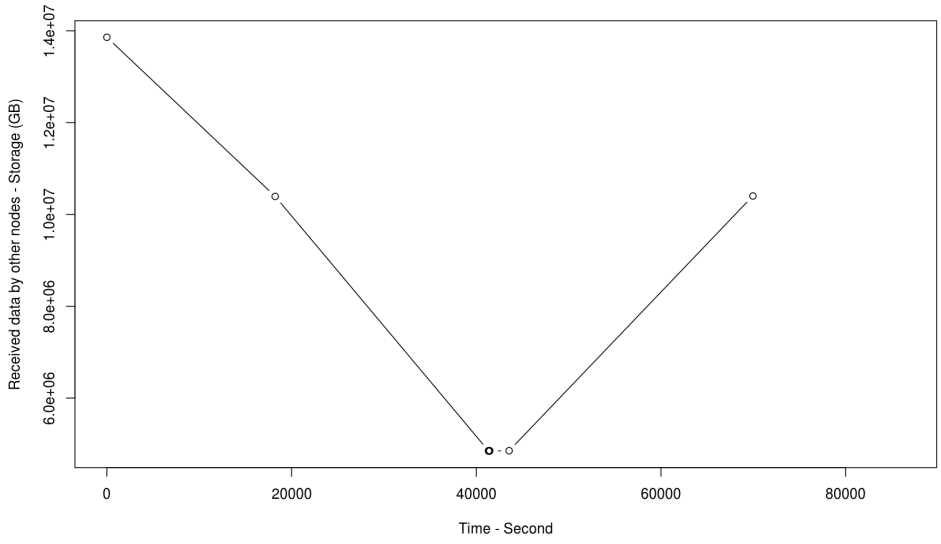

Figure F.134: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.05 and a growth factor $f = 2$.

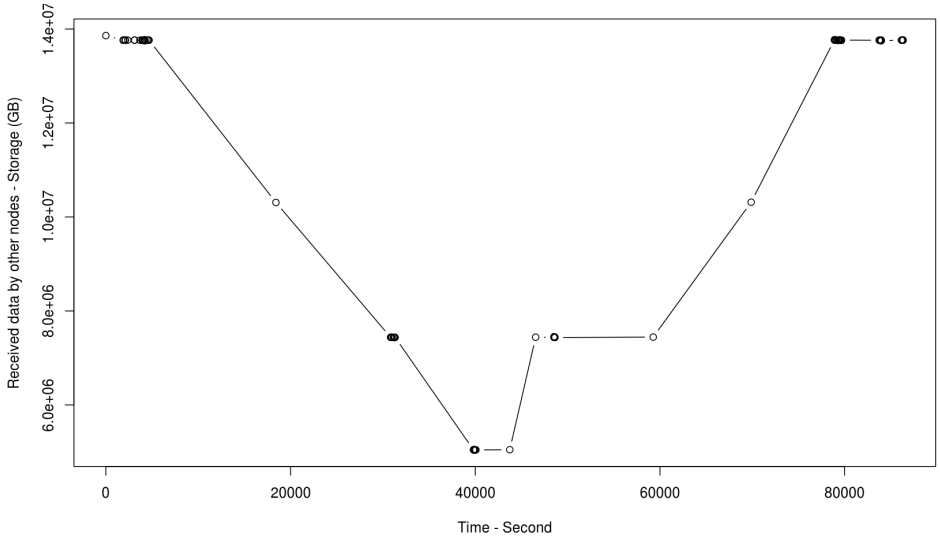

Figure F.135: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor $\beta = 0.1$ and a growth factor $f = 1.2$.

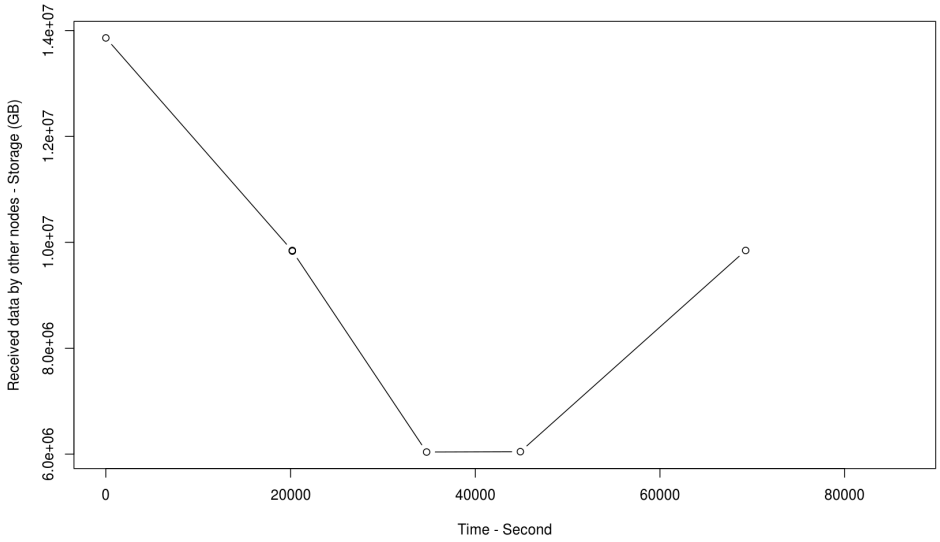

Figure F.136: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor $\beta = 0.1$ and a growth factor $f = 1.4$.

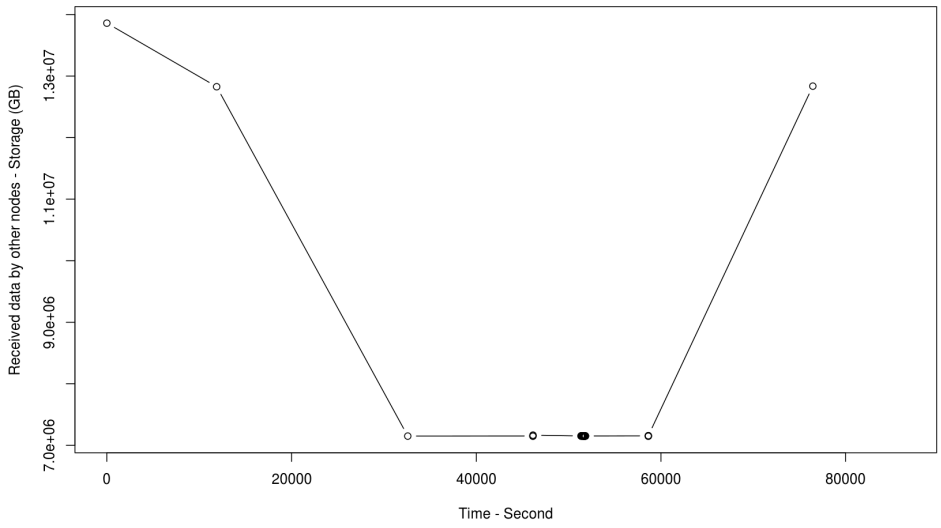

Figure F.137: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.1 and a growth factor f = 1.6.

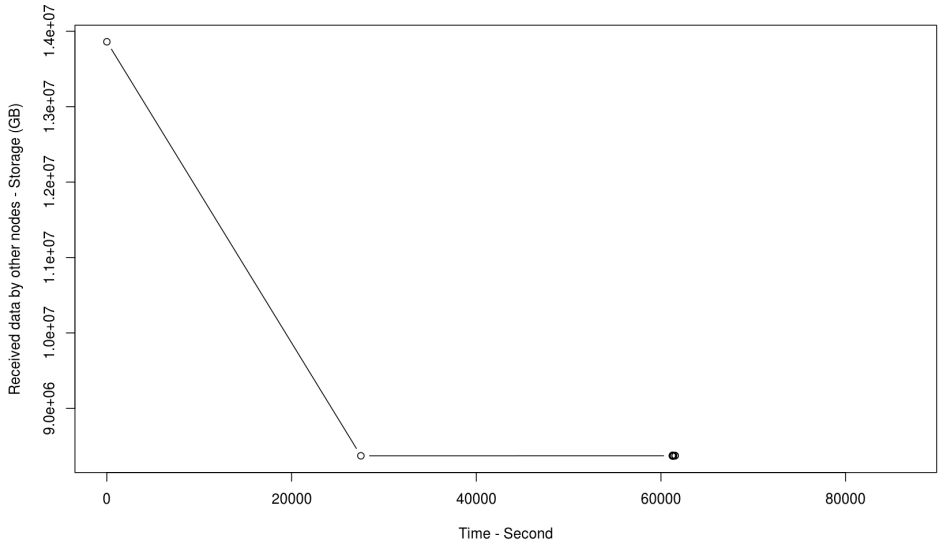

Figure F.138: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor β = 0.1 and a growth factor f = 1.8.

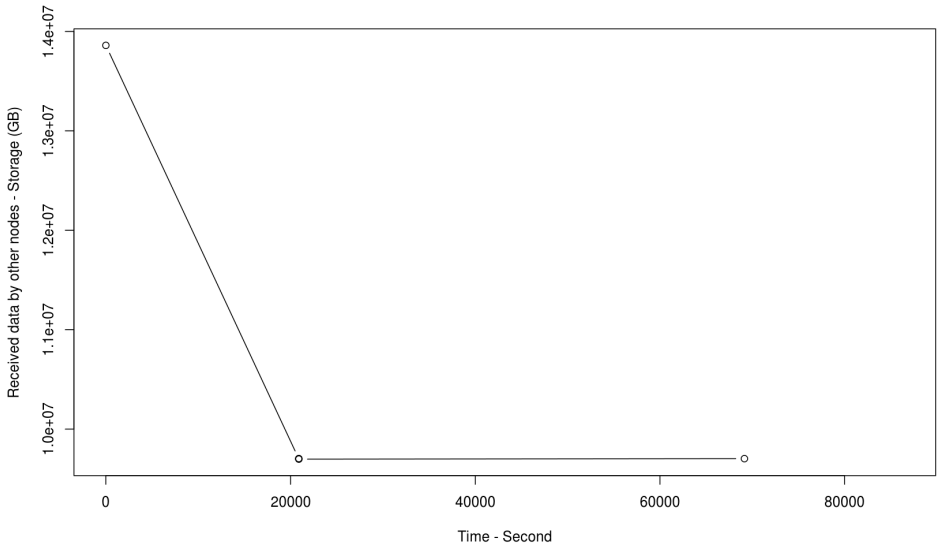

Figure F.139: Exponential-sized class-based update policy. How a cloud management system views data center's *storage* capacity when a base factor $\beta = 0.1$ and a growth factor $f = 2$.

Appendix G

Absolute threshold-based update policy experiments

This Appendix provides complete set of results when an **absolute threshold-based update policy** with different threshold values was applied on a proposed simulated data center's resources (see section 5.4.1). It is worth mentioning that the points that shown in circle in these graphs depict the time that the link-state update is sent out into the network. These Figures provide an overview to help understand how other node(s) (e.g., CMS) in the network views the data center's resources based on each threshold value.

Additionally, information about the number of updates per different threshold value based on a simulated data center's resource changes (i.e., CPU, RAM, and storage) are provided in this Appendix.

G.1 Data center's CPU

Table G.1: Absolute threshold-based update policy. Number of Cloud LSA updates based on changes in proposed data center's **CPU**. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
0.01	32364	32415	32313	31478	32878	255.160
0.02	23368	23410	23327	22821	23802	209.775
0.04	13453	13482	13424	13152	13799	146.225
0.06	8561	8584	8538	8283	8851	117.986
0.08	5788	5806	5770	5586	5993	89.545
0.1	4238	4254	4222	4078	4469	79.131
0.5	472	476	469	429	517	18.803
1	211	213	209	188	232	10.203
2	100	101	99	87	110	5.076
3	65	66	65	56	73	3.729
4	48	49	48	40	55	2.882
5	38	38	38	32	43	2.285
10	18	18	18	16	21	1.118

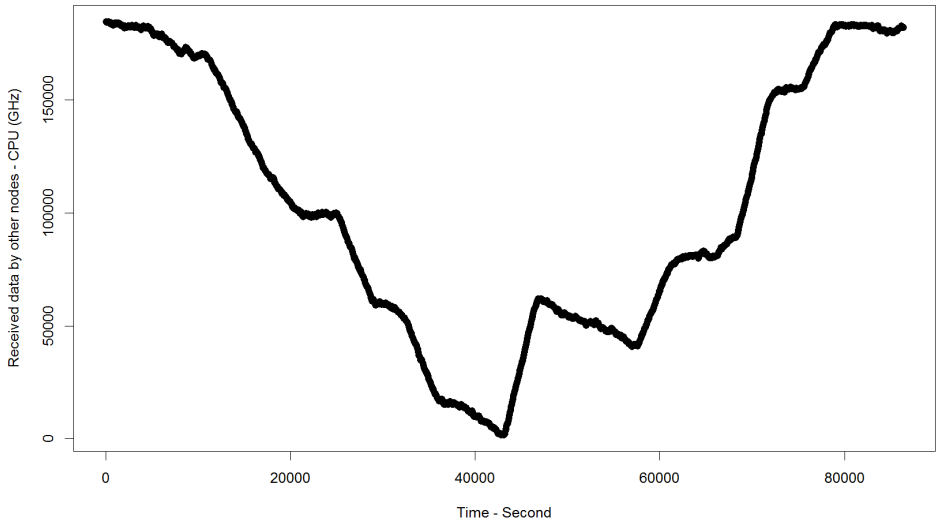

Figure G.1: Data center sample CPU (GHz) capacity per second

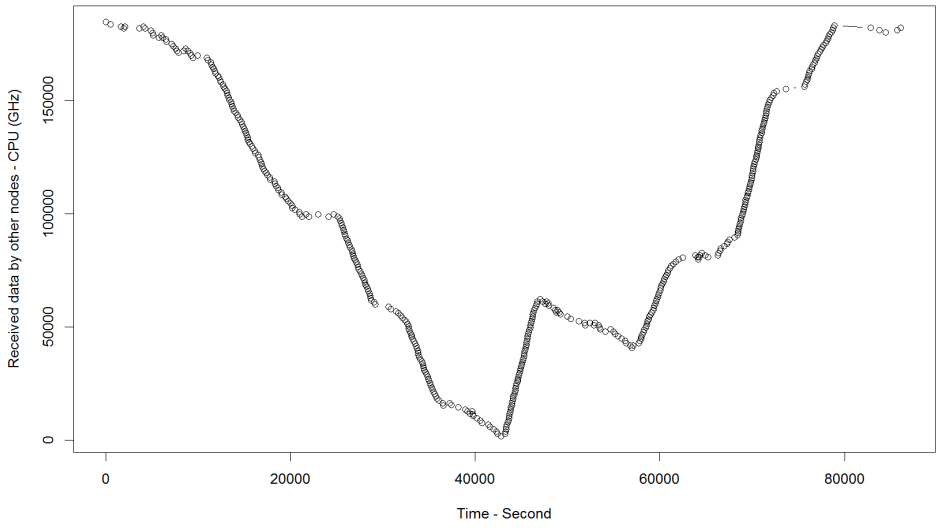

Figure G.2: Absolute threshold-based update policy. How a cloud management system views Storage resources with an *absolute* threshold value 0.5%, the updates points are plotted a circles.

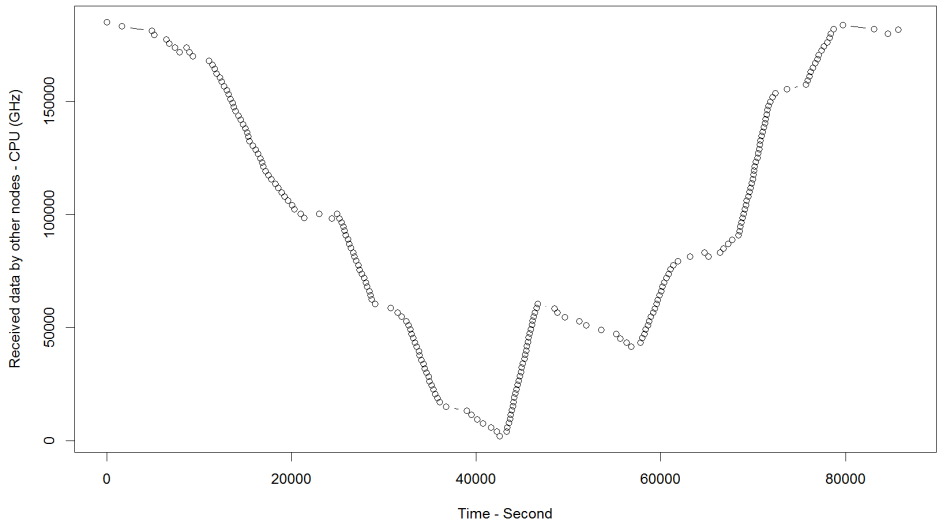

Figure G.3: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 1%, the updates points are plotted a circles.

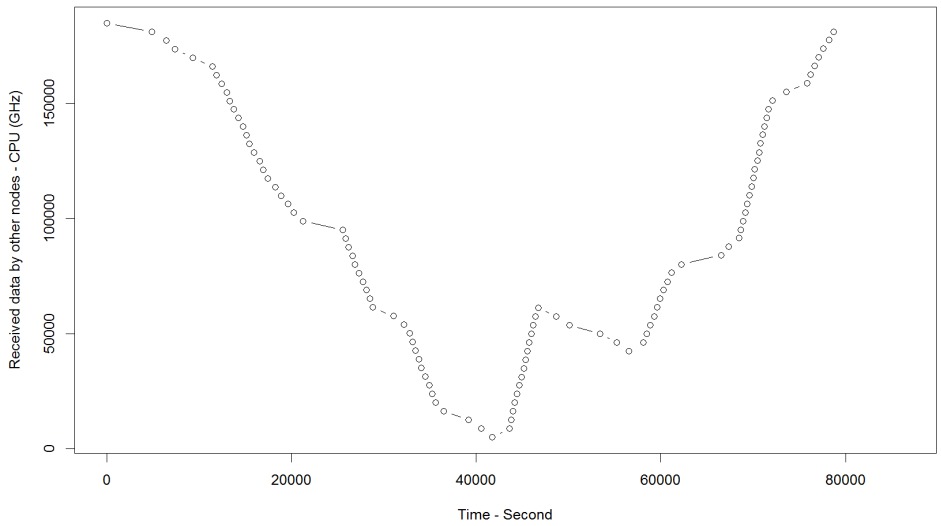

Figure G.4: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 2%, the updates points are plotted a circles.

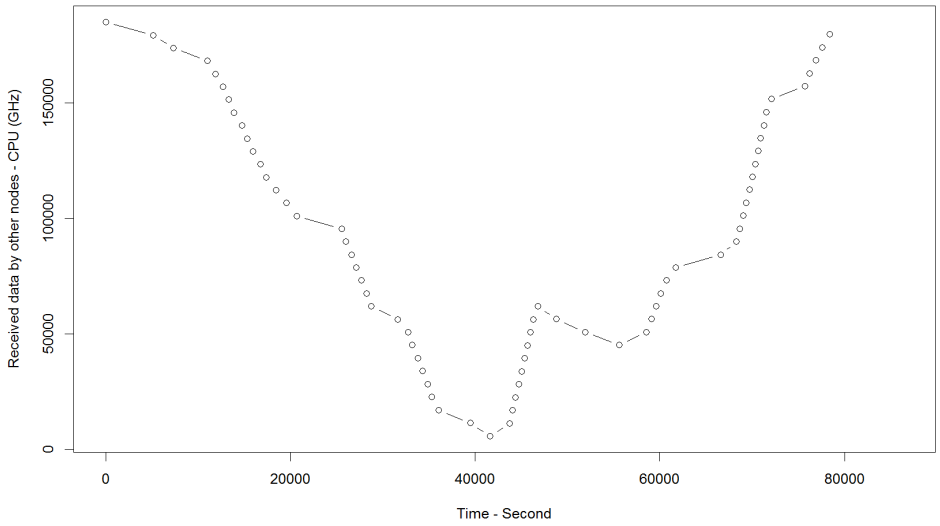

Figure G.5: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 3%, the updates points are plotted a circles.

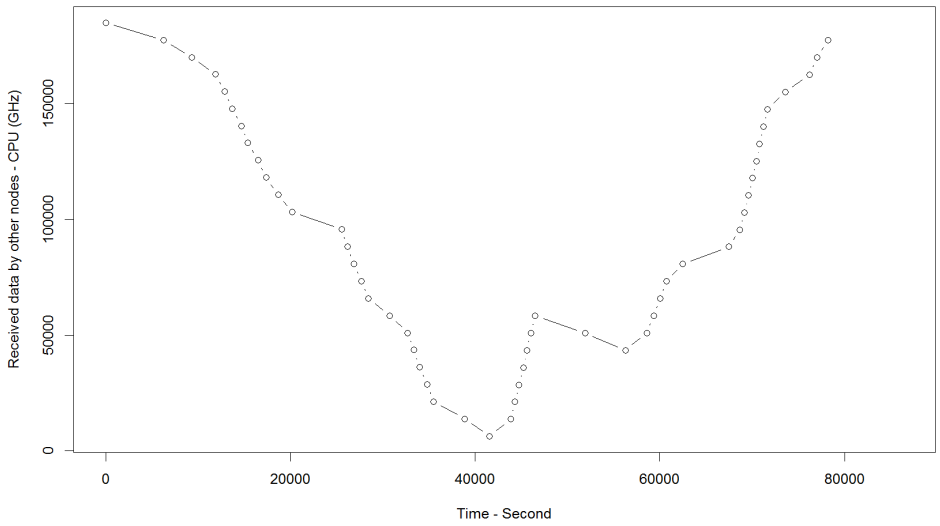

Figure G.6: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 4%, the updates points are plotted a circles.

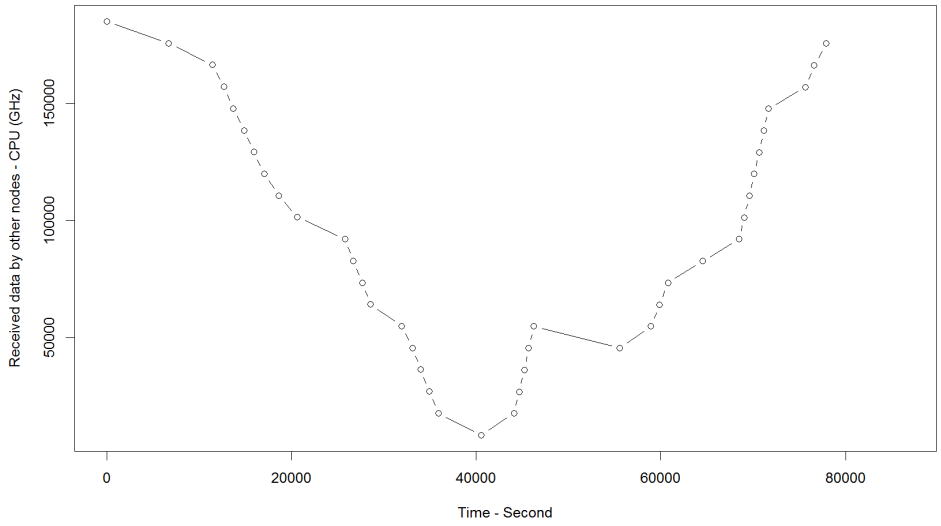

Figure G.7: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 5%, the updates points are plotted a circles.

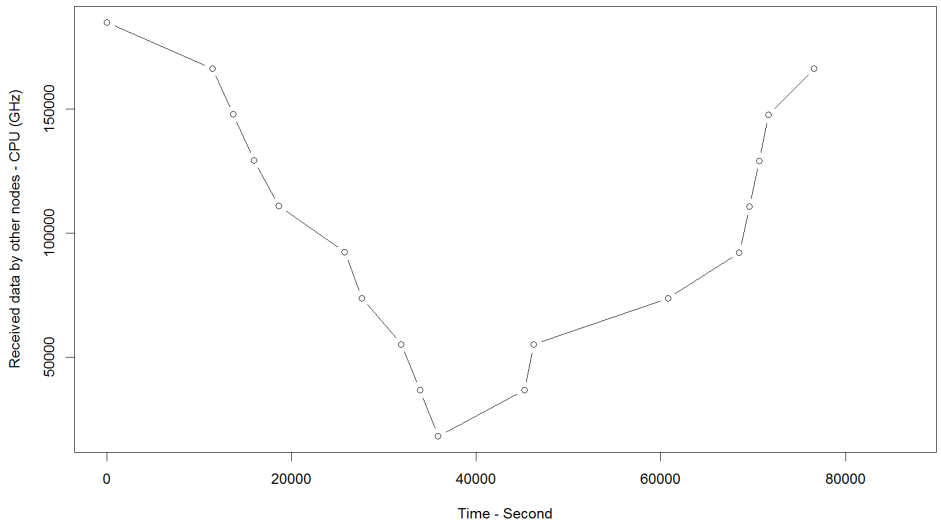

Figure G.8: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 10%, the updates points are plotted a circles.

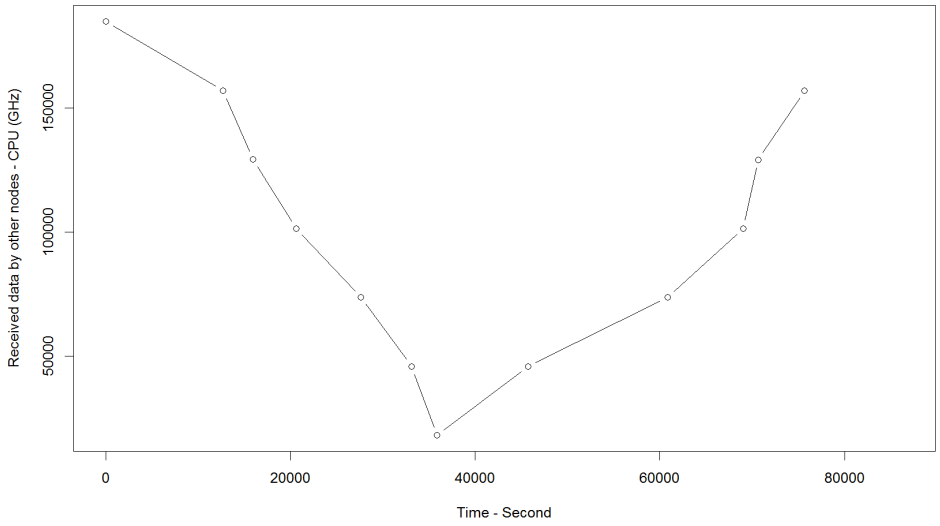

Figure G.9: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 15%, the updates points are plotted a circles.

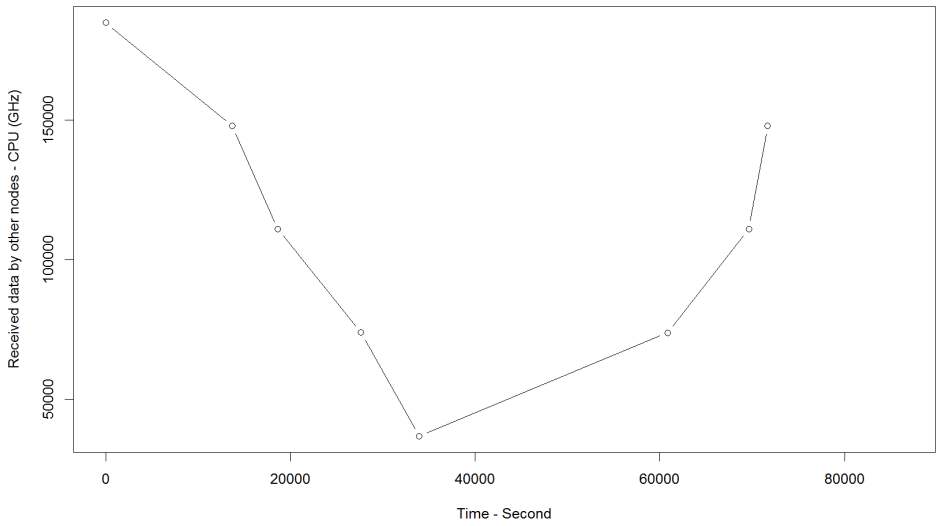

Figure G.10: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 20%, the updates points are plotted a circles.

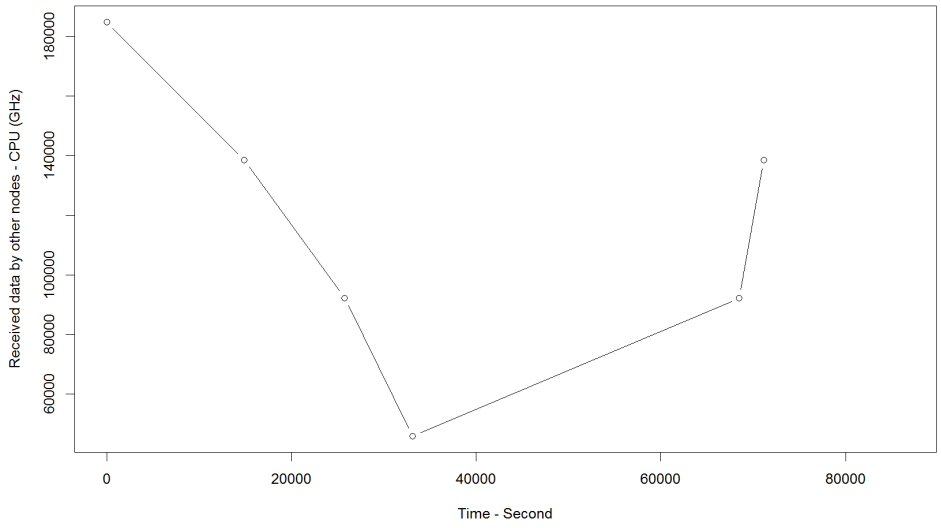

Figure G.11: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 25%, the updates points are plotted a circles.

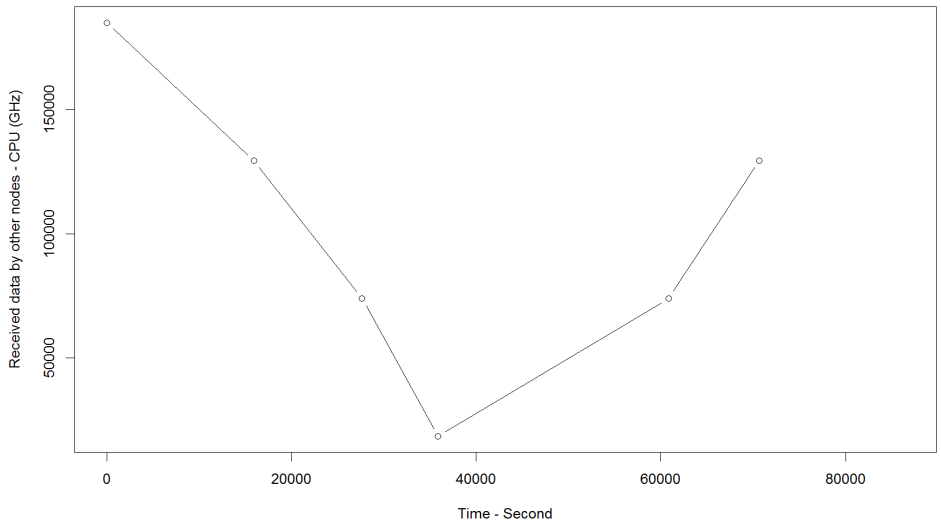

Figure G.12: Absolute threshold-based update policy. How a cloud management system views CPU resources with an *absolute* threshold value 30%, the updates points are plotted a circles.

G.2 Data center's RAM

Table G.2: Absolute threshold-based update policy. Number of Cloud LSA updates based on changes in proposed data center's **RAM**. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
0.01	33044	33095	32992	32125	33576	260.433
0.02	24934	24976	24891	24362	25350	211.849
0.04	14136	14166	14105	13806	14558	154.878
0.06	8507	8530	8485	8179	8754	115.097
0.08	5698	5716	5679	5460	5896	93.819
0.1	4141	4156	4125	3880	4294	78.694
0.5	467	470	463	424	506	19.251
1	211	214	209	188	232	10.239
2	101	102	100	89	112	5.662
3	66	67	65	56	75	3.927
4	49	49	48	42	55	2.869
5	38	39	38	34	43	2.363
10	18	18	18	16	21	1.118

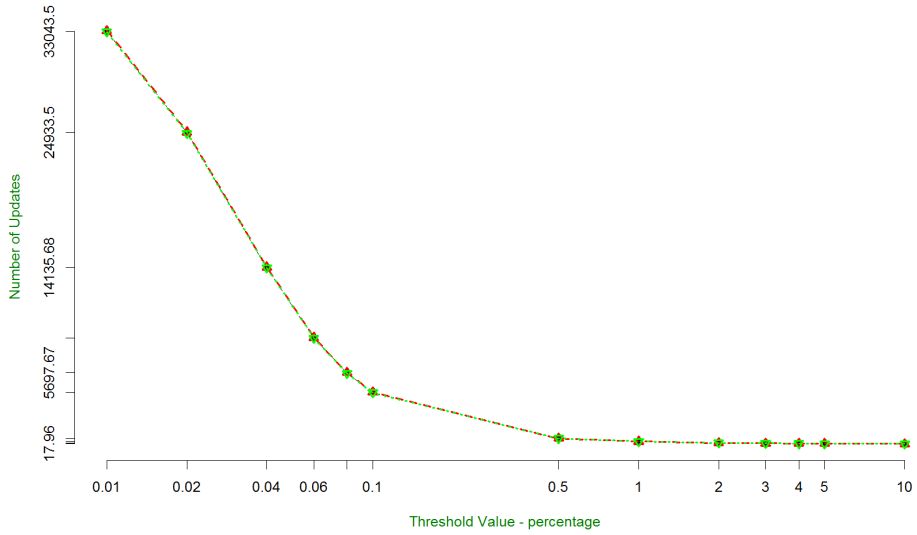

Figure G.13: *Relative threshold-based* update policy. Number of updates per different threshold values based on changes of a data center's RAM capacity. The "X" axis are in logarithm scale.

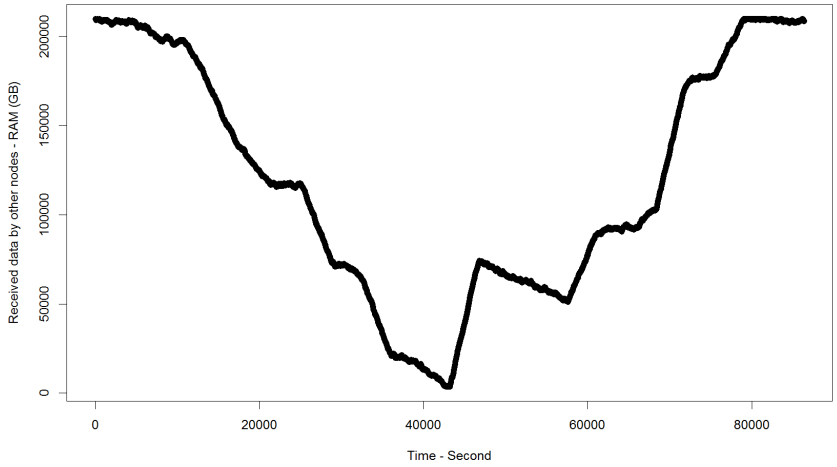

Figure G.14: Data center sample RAM (GB) capacity per second

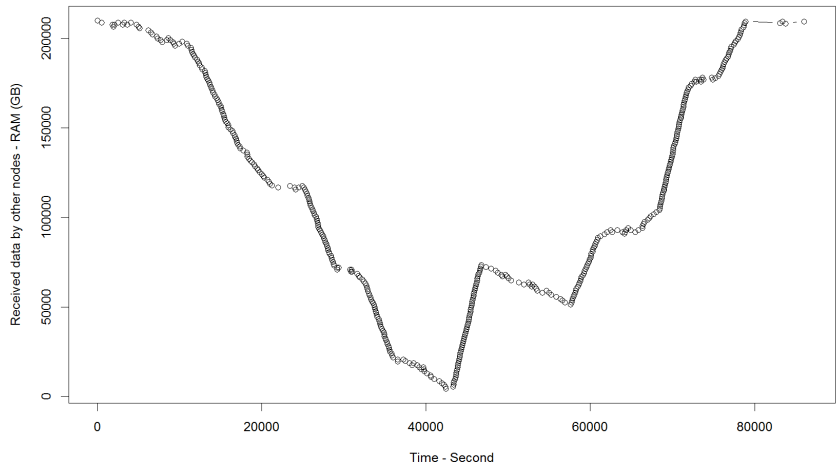

Figure G.15: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 0.5%, the updates points are plotted a circles.

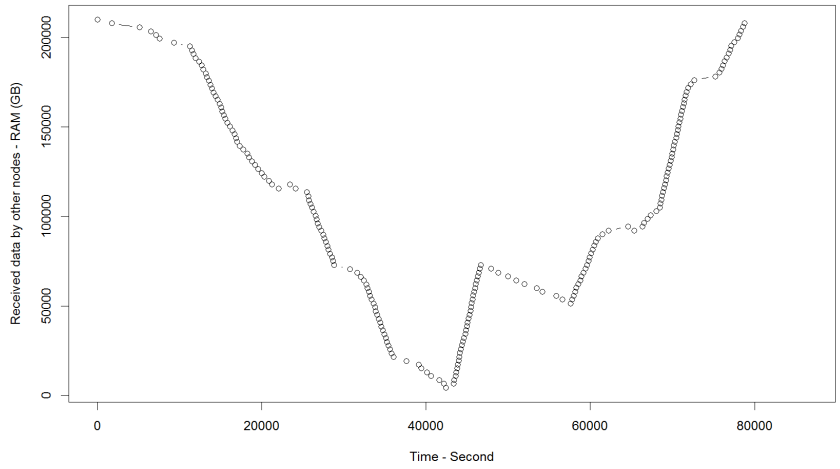

Figure G.16: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 1%, the updates points are plotted a circles.

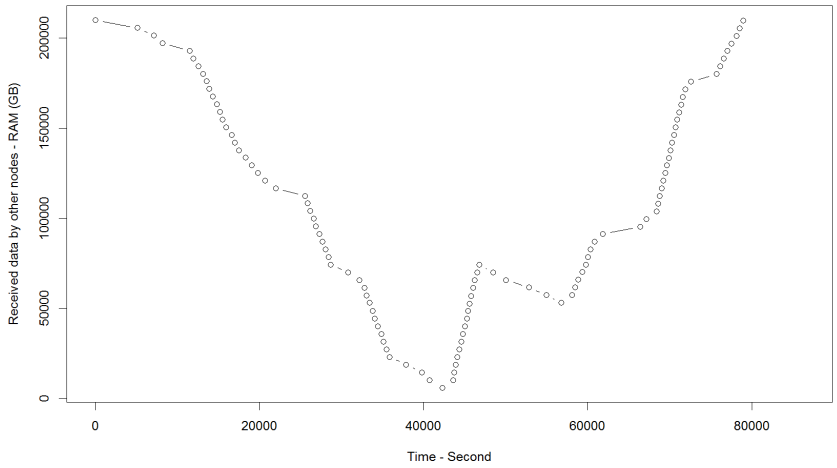

Figure G.17: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 2%, the updates points are plotted a circles.

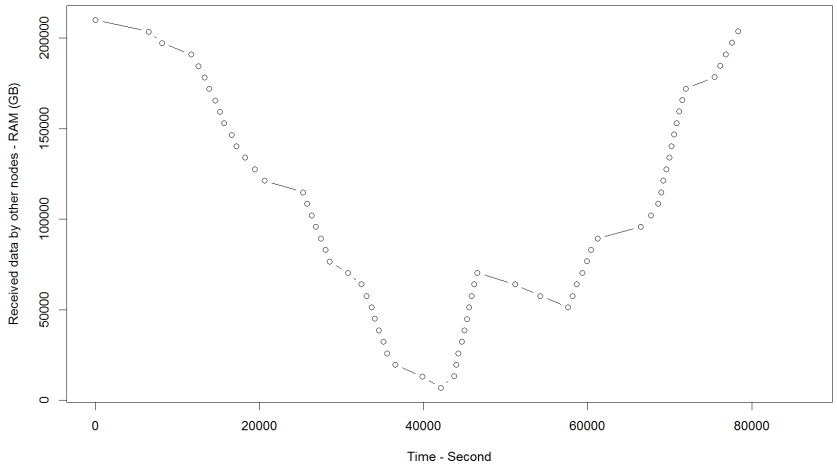

Figure G.18: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 3%, the updates points are plotted a circles.

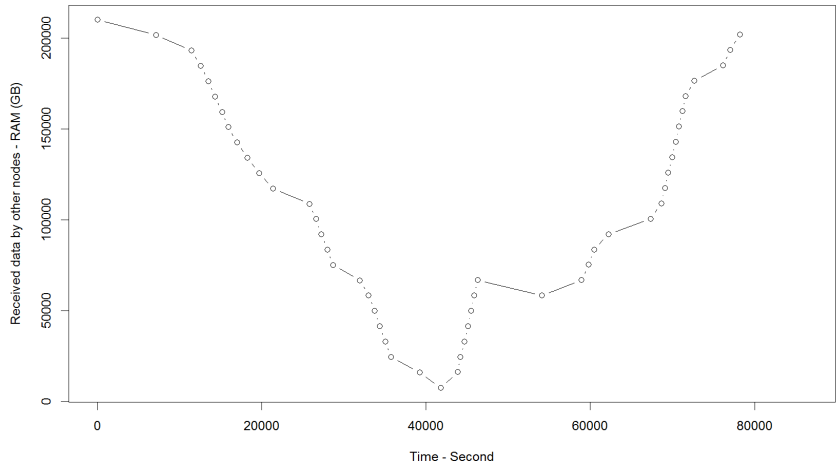

Figure G.19: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 4%, the updates points are plotted a circles.

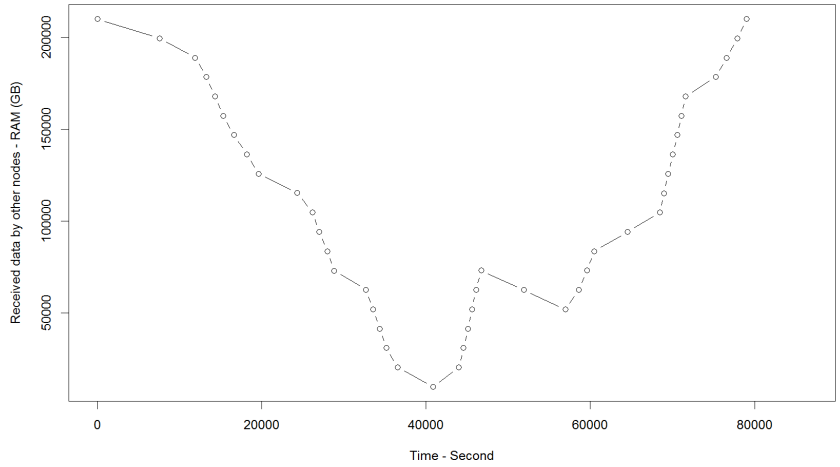

Figure G.20: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 5%, the updates points are plotted a circles.

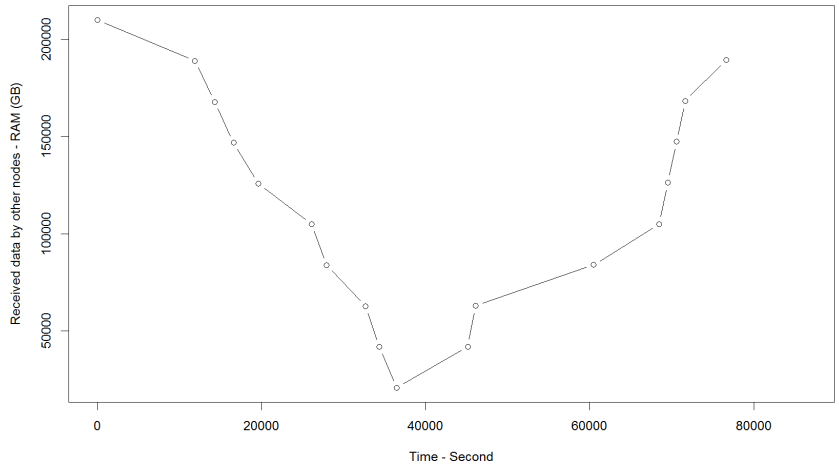

Figure G.21: Absolute threshold-based update policy. How a cloud management system views RAM resources with an *absolute* threshold value 10%, the updates points are plotted a circles.

G.3 Data center's storage

Table G.3: Absolute threshold-based update policy. Number of Cloud LSA updates based on changes in proposed data center's **storage**. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
0.01	30454	30504	30403	29618	31040	254.096
0.02	19158	19196	19120	18691	19624	191.118
0.04	8652	8674	8631	8423	8860	109.898
0.06	4914	4931	4897	4729	5109	85.013
0.08	3244	3257	3231	3095	3379	66.151
0.1	2352	2362	2341	2222	2466	54.955
0.5	297	299	294	262	323	13.335
1	139	141	138	120	155	7.287
2	67	68	66	60	75	3.594
3	44	45	44	36	51	2.975
4	32	33	32	26	37	2.090
5	25	25	25	20	29	1.954
10	12	12	12	10	13	0.924

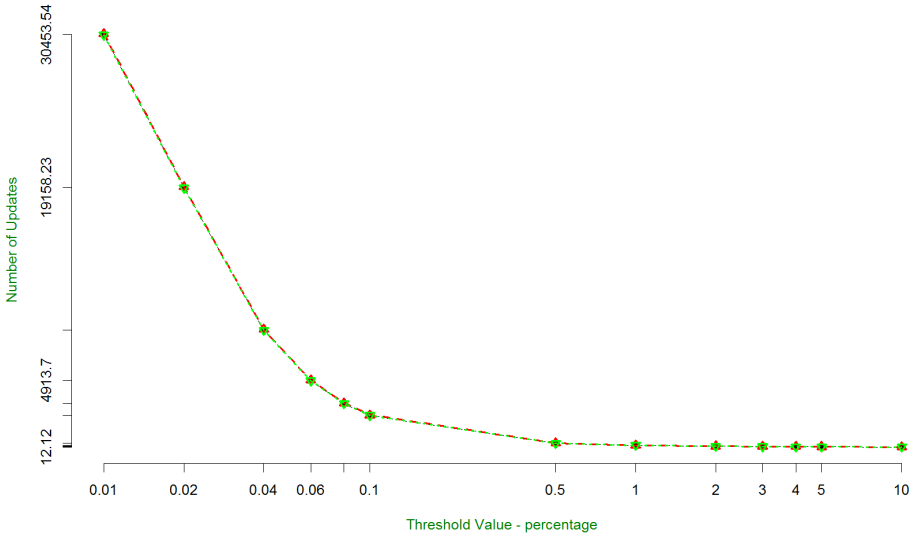

Figure G.22: *Absolute threshold-based* update policy. Number of updates per different threshold values based on changes of a data center's storage capacity. The "X" axis are in logarithm scale.

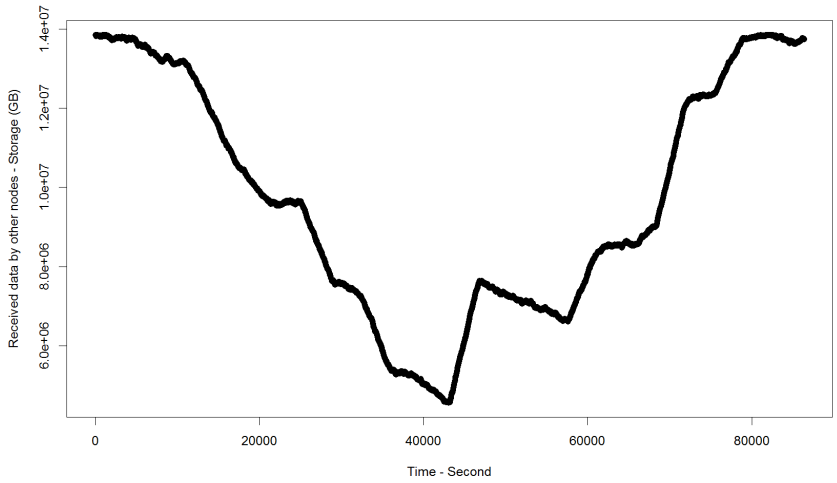

Figure G.23: Data center sample storage (GB) capacity per second.

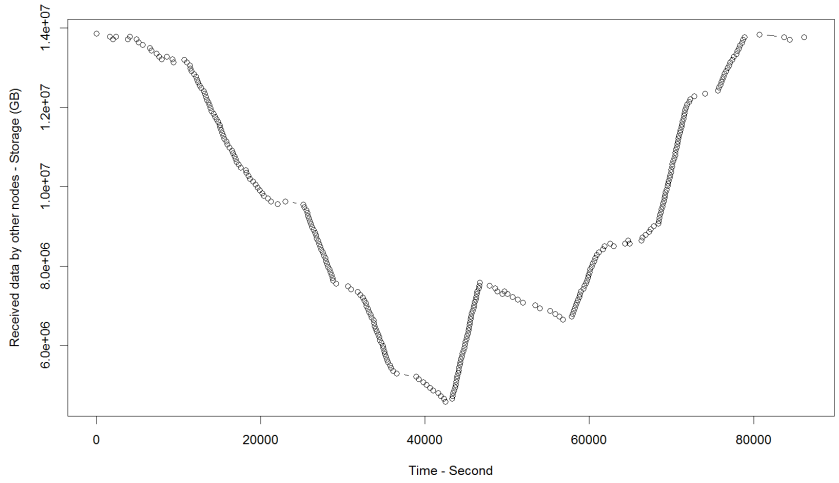

Figure G.24: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 0.5%, the updates points are plotted a circles.

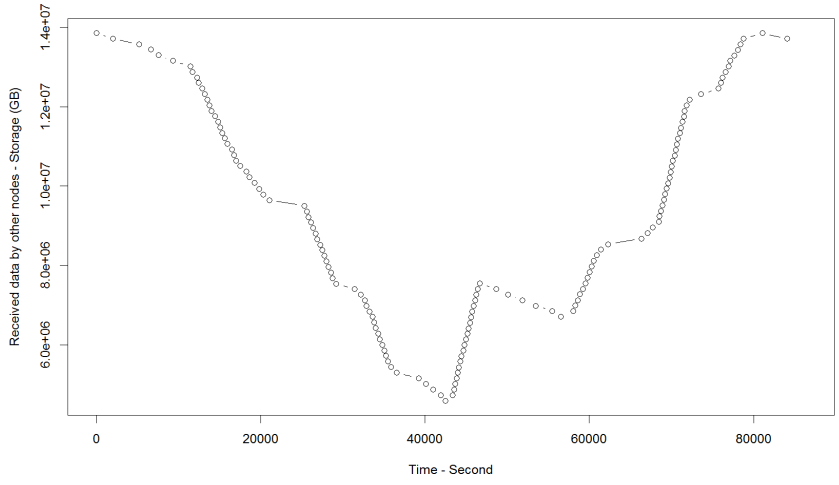

Figure G.25: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 1%, the updates points are plotted a circles.

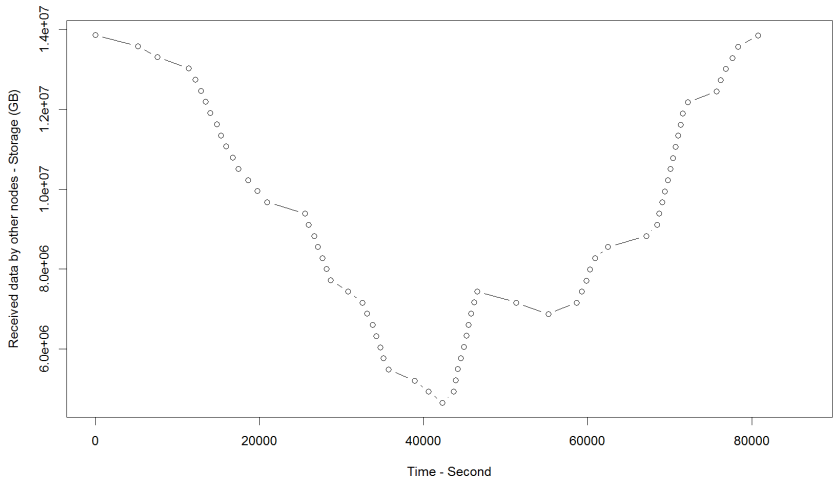

Figure G.26: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 2%, the updates points are plotted a circles.

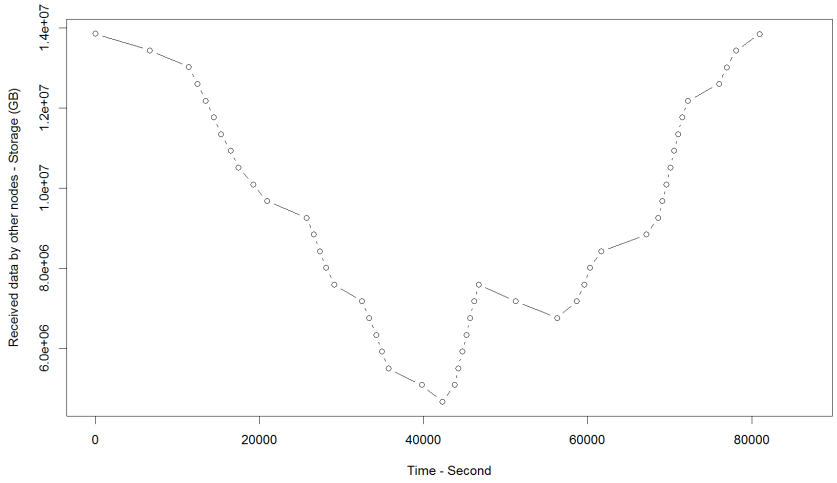

Figure G.27: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 3%, the updates points are plotted a circles.

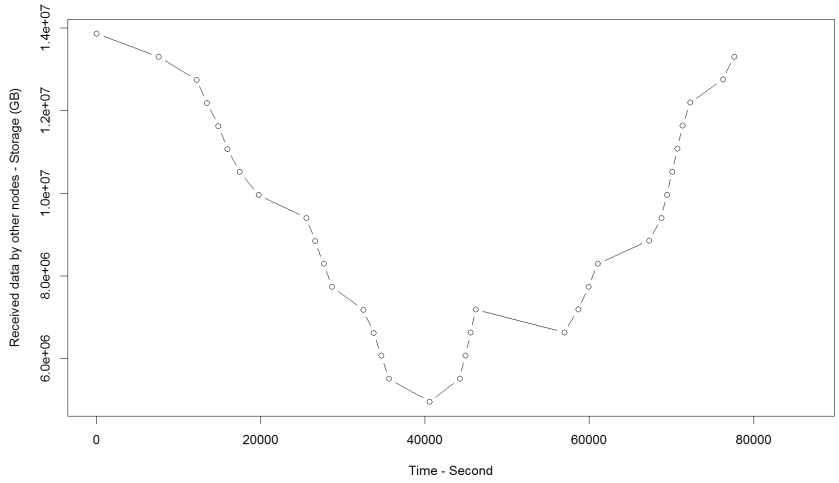

Figure G.28: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 4%, the updates points are plotted a circles.

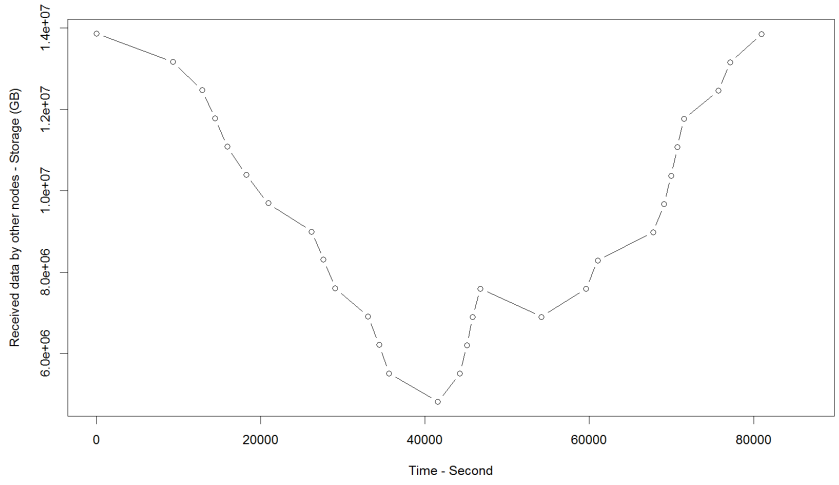

Figure G.29: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 5%, the updates points are plotted a circles.

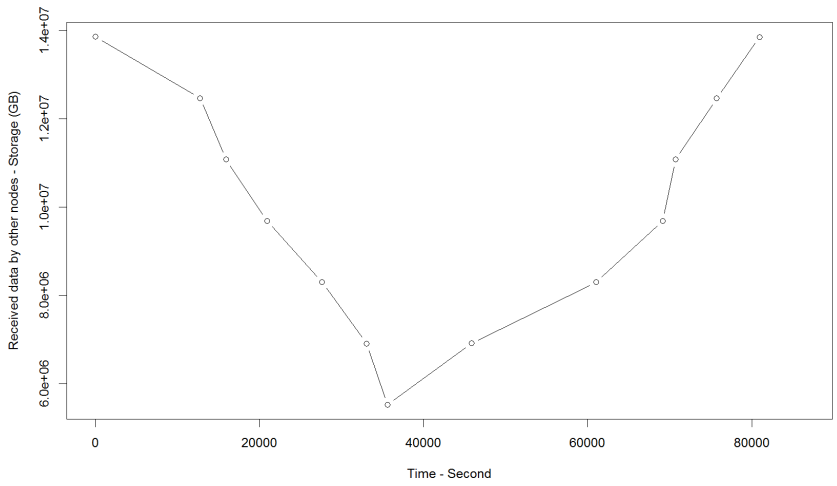

Figure G.30: Absolute threshold-based update policy. How a cloud management system views storage resources with an *absolute* threshold value 10%, the updates points are plotted a circles.

Appendix H

Relative threshold update policy experiments

This Appendix provides complete set of results when a relative threshold update policy with different threshold values (e.g., 5, 10, 15, 20, and etc) was applied on a proposed simulated data center's resources (see section 5.4.2). It is worth mentioning that the points that shown in circle in these graphs depict the time that the link-state update is sent out into the network. These Figures provide an overview to help understand how other node(s) in the network views the data center's resources based on each threshold value.

Additionally, information about the number of updates per different threshold value based on a simulated data center's resource changes (i.e., CPU, RAM, and storage) are provided in this Appendix.

H.1 Data center's CPU

Table H.1: Relative threshold update policy. Number of Cloud LSA updates based on changes in a proposed data center's **CPU**. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
5	140	153	126	72	539	67.814
10	66	73	60	32	285	33.282
15	43	47	39	22	173	21.422
20	31	35	28	16	143	16.955
25	25	27	22	12	104	12.846
30	20	22	18	10	92	10.996
35	17	19	15	8	80	9.599
40	15	16	13	8	72	8.501
45	13	14	11	6	64	7.679
50	12	13	10	6	54	6.786
55	10	11	9	6	48	6.142
60	9	10	8	3	44	5.767
65	9	10	8	4	40	5.160
70	7	8	6	4	36	4.803
75	6	7	6	4	29	4.381
80	6	7	5	4	30	4.282
85	6	7	5	1	29	3.874
90	5	5	4	1	26	4.021
95	3	4	2	1	24	4.147

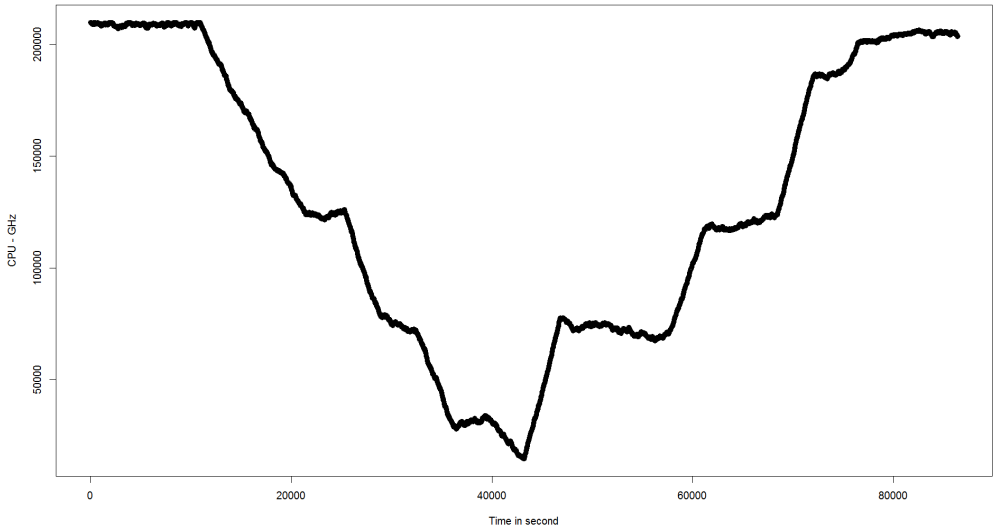

Figure H.1: Data center sample CPU (GHz) capacity per second

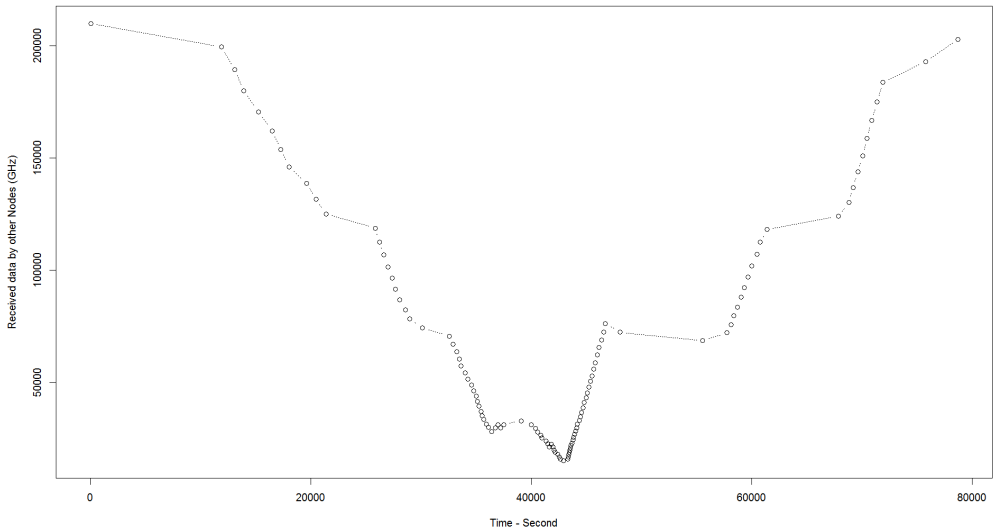

Figure H.2: How cloud management system views CPU resources with a *relative* threshold value 5, the updates points are plotted a circles.

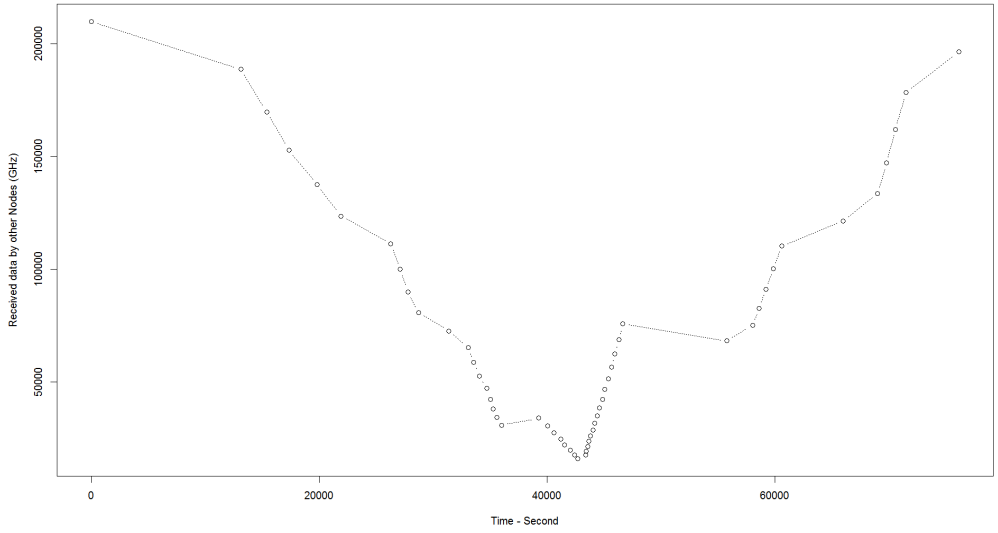

Figure H.3: How cloud management system views CPU resources with a *relative* threshold value 10, the updates points are plotted a circles.

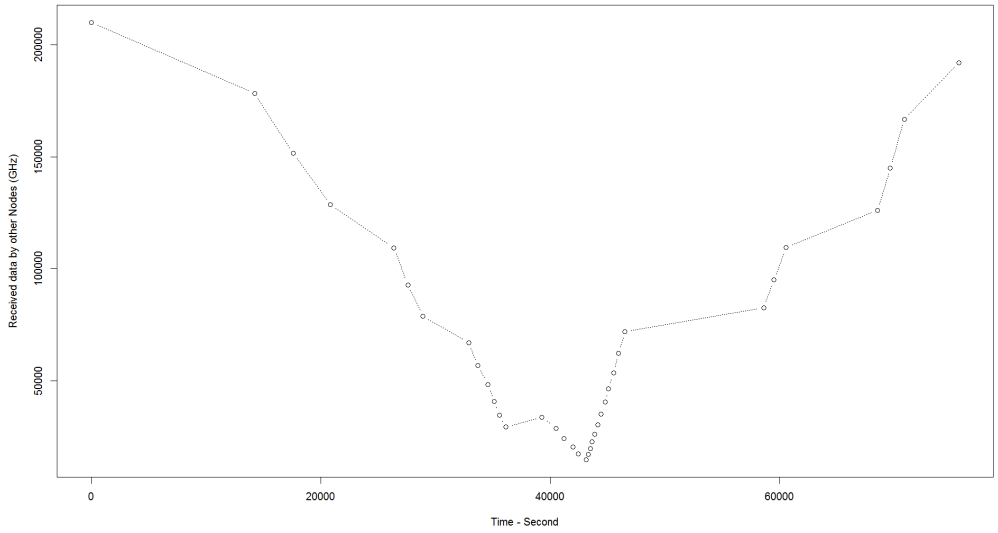

Figure H.4: How cloud management system views CPU resources with a *relative* threshold value 15, the updates points are plotted a circles.

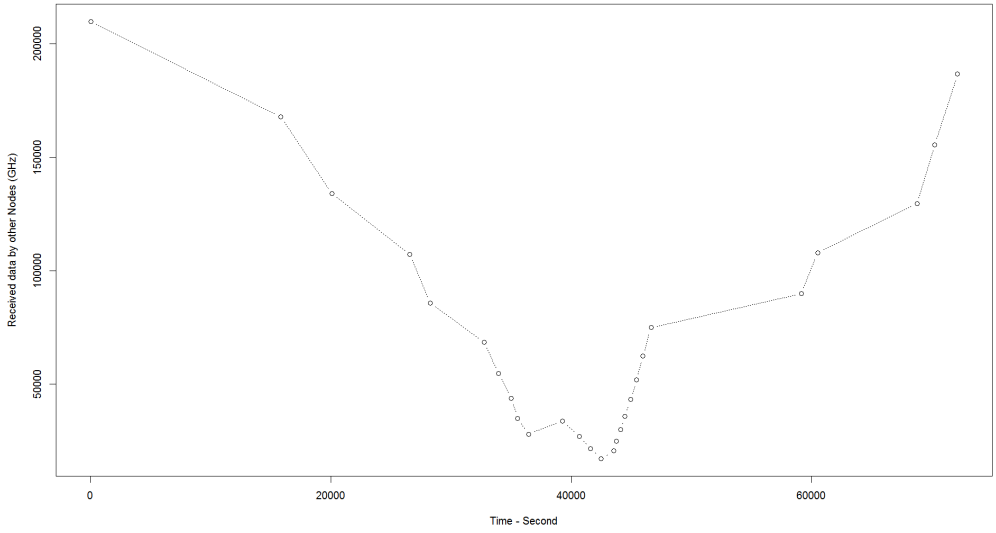

Figure H.5: How cloud management system views CPU resources with a *relative* threshold value 20, the updates points are plotted a circles.

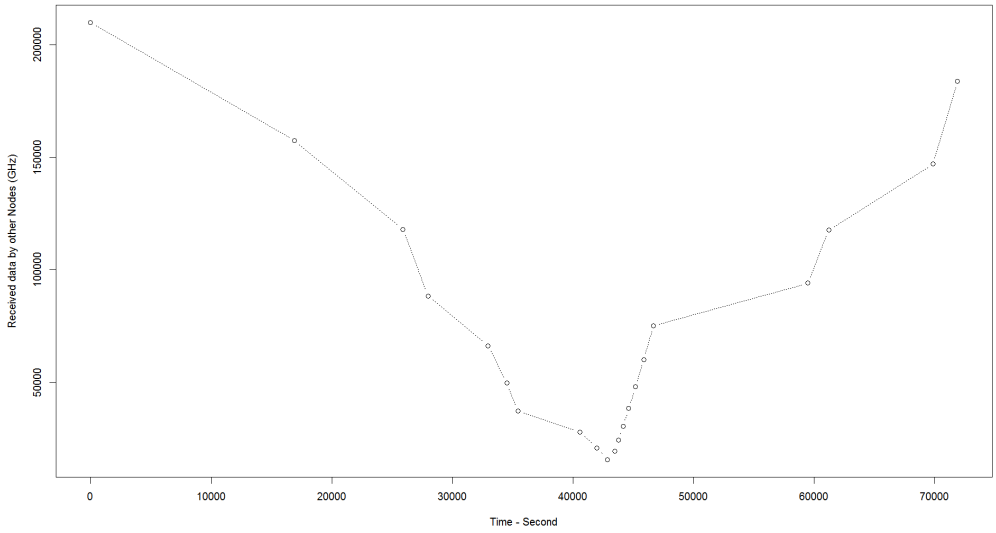

Figure H.6: How cloud management system views CPU resources with a *relative* threshold value 25, the updates points are plotted a circles.

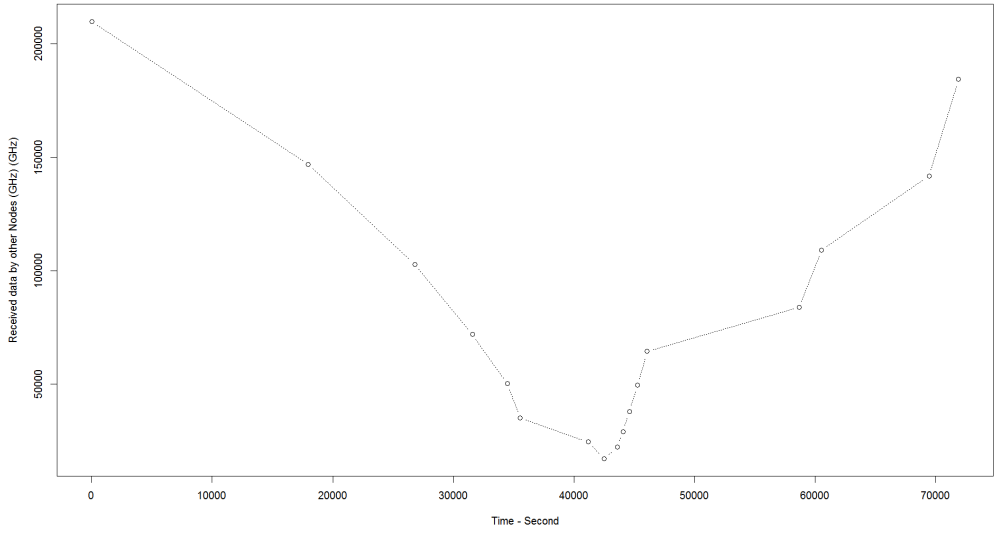

Figure H.7: How cloud management system views CPU resources with a *relative* threshold value 30, the updates points are plotted a circles.

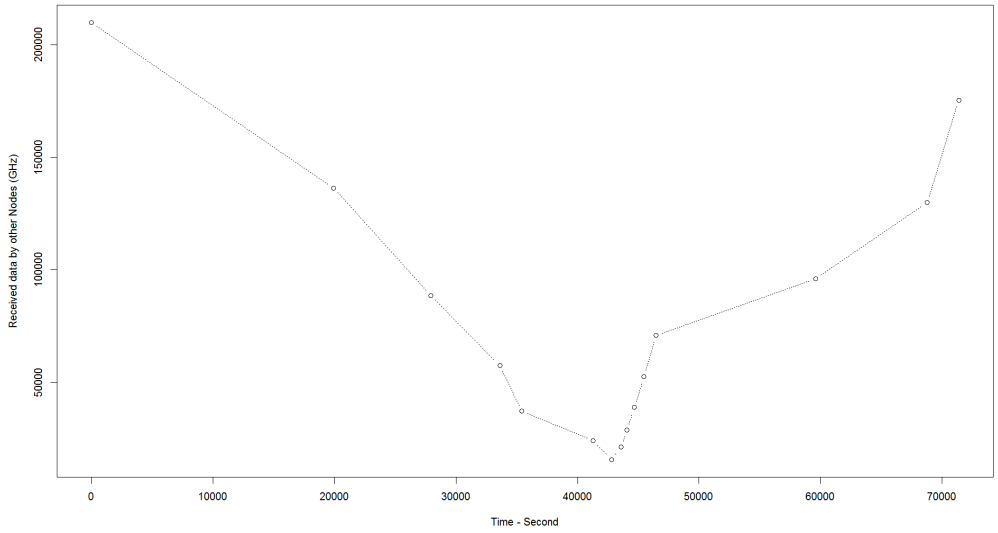

Figure H.8: How cloud management system views CPU resources with a *relative* threshold value 35, the updates points are plotted a circles.

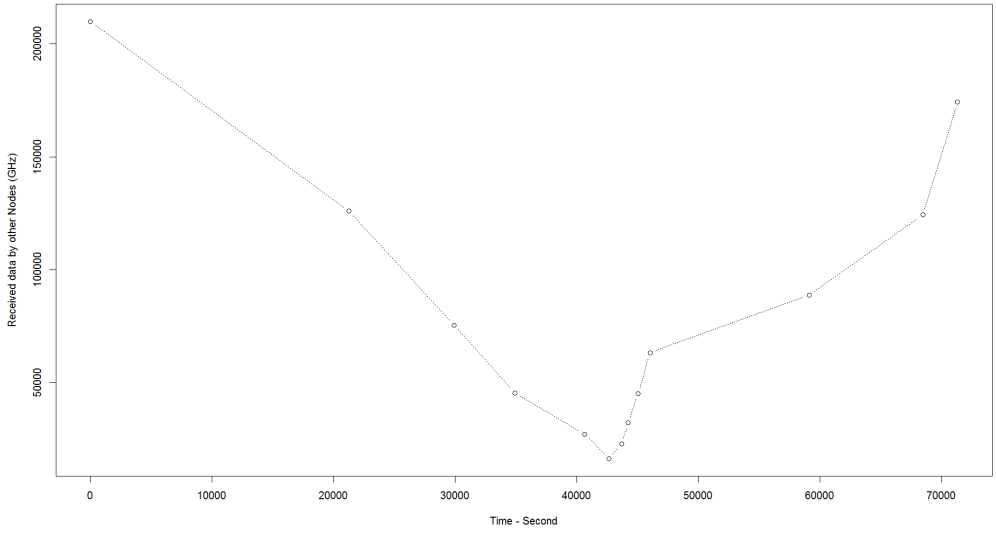

Figure H.9: How cloud management system views CPU resources with a *relative* threshold value 40, the updates points are plotted a circles.

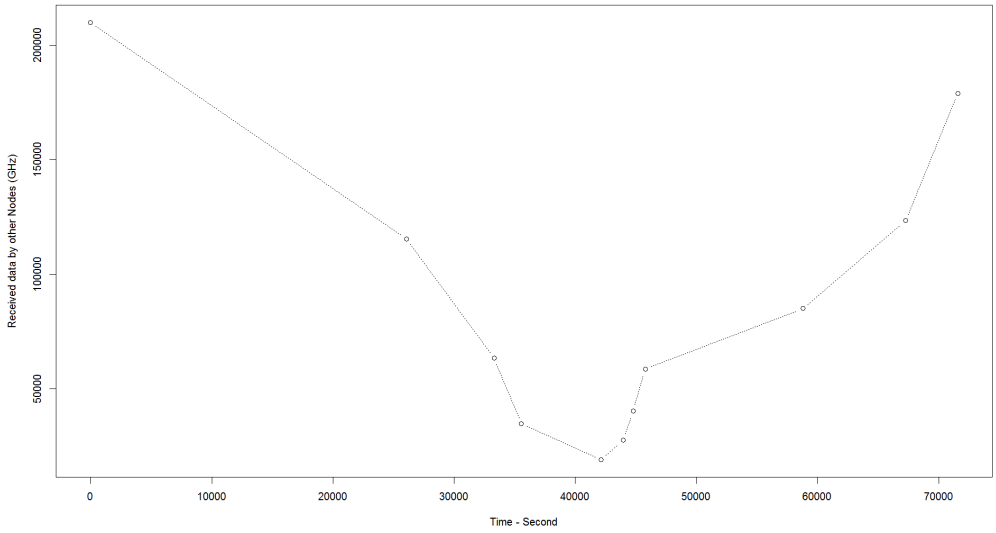

Figure H.10: How cloud management system views CPU resources with a *relative* threshold value 45, the updates points are plotted a circles.

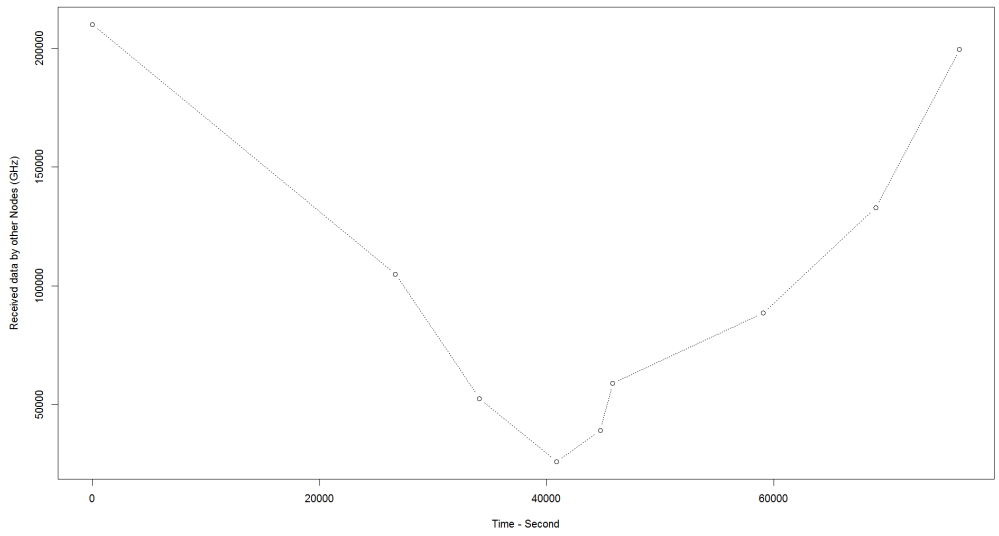

Figure H.11: How cloud management system views CPU resources with a *relative* threshold value 50, the updates points are plotted a circles.

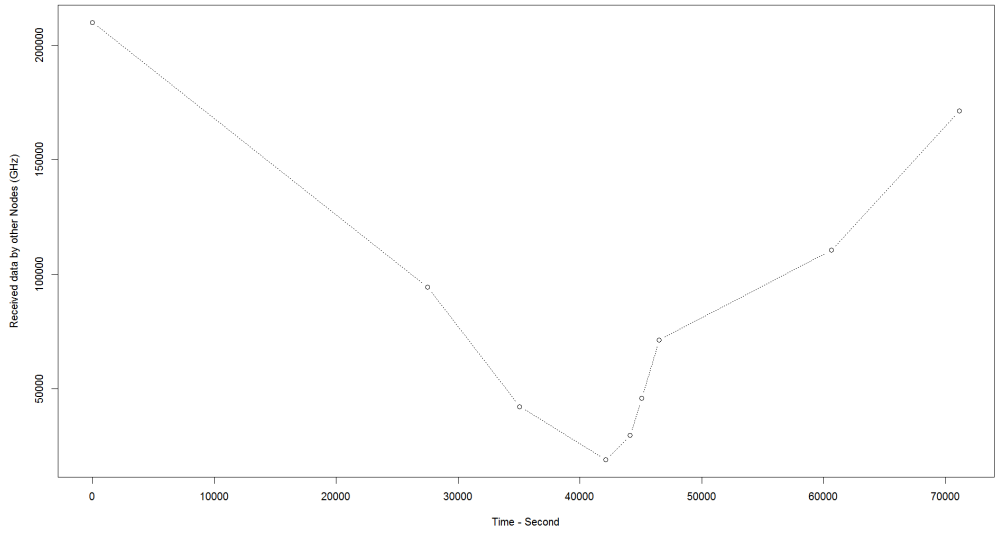

Figure H.12: How cloud management system views CPU resources with a *relative* threshold value 55, the updates points are plotted a circles.

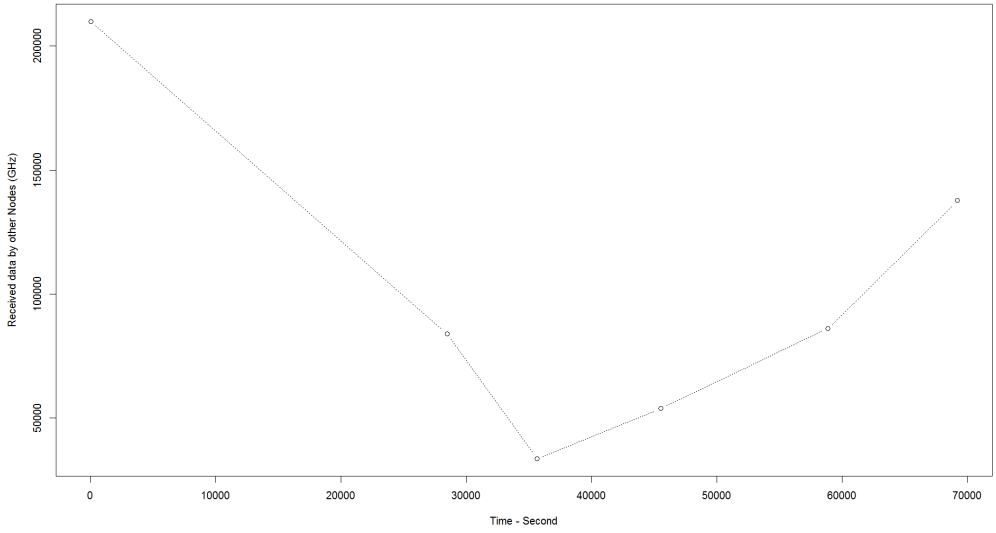

Figure H.13: How cloud management system views CPU resources with a *relative* threshold value 60, the updates points are plotted a circles.

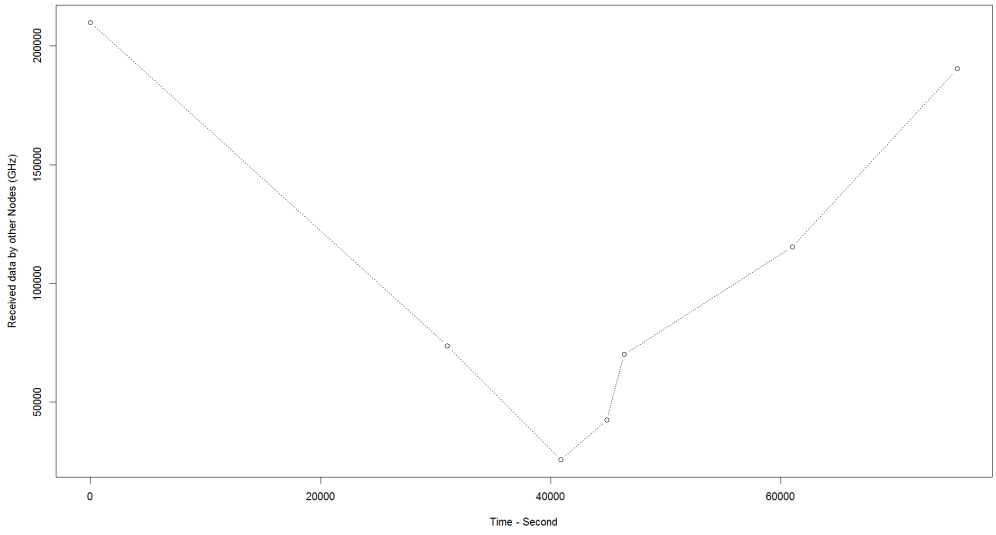

Figure H.14: How cloud management system views CPU resources with a *relative* threshold value 65, the updates points are plotted a circles.

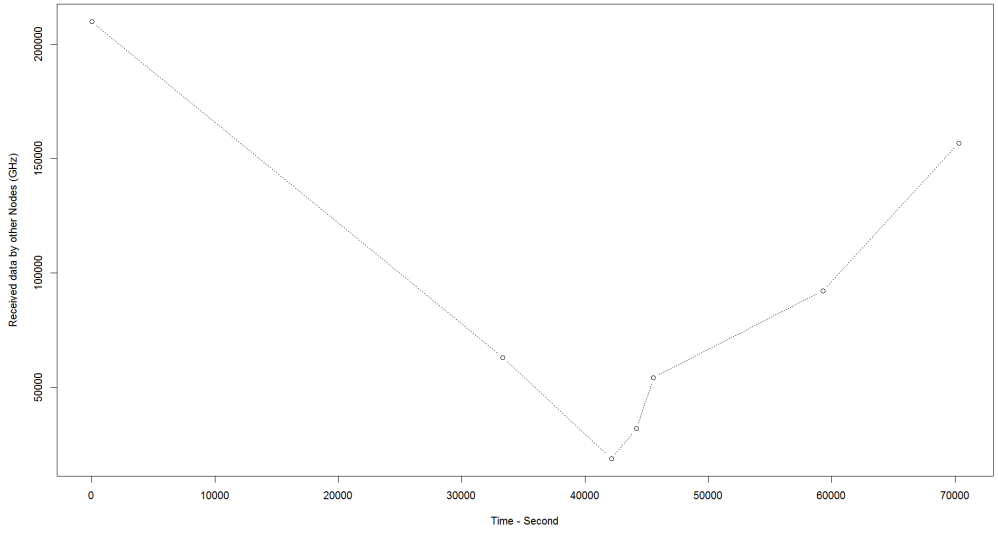

Figure H.15: How cloud management system views CPU resources with a *relative* threshold value 70, the updates points are plotted a circles.

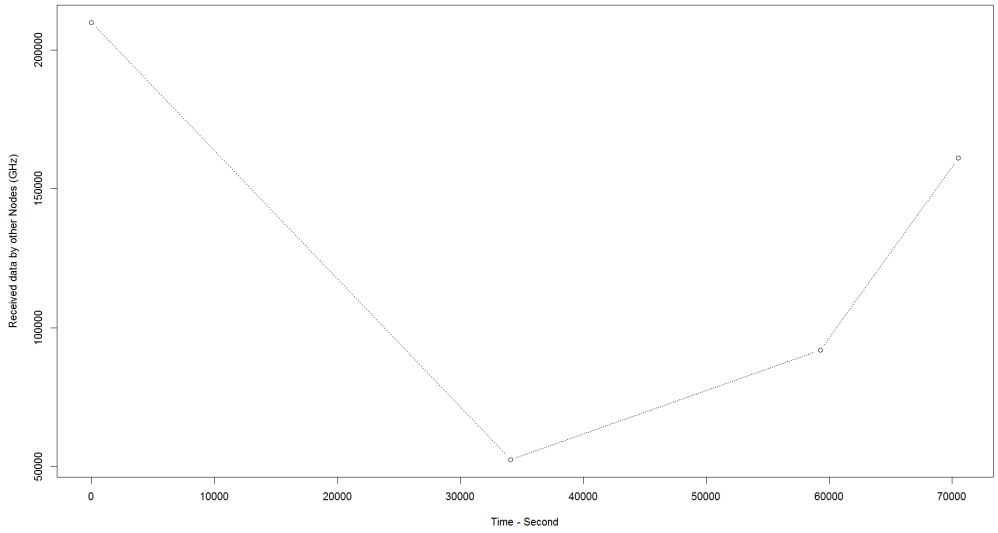

Figure H.16: How cloud management system views CPU resources with a *relative* threshold value 75, the updates points are plotted a circles.

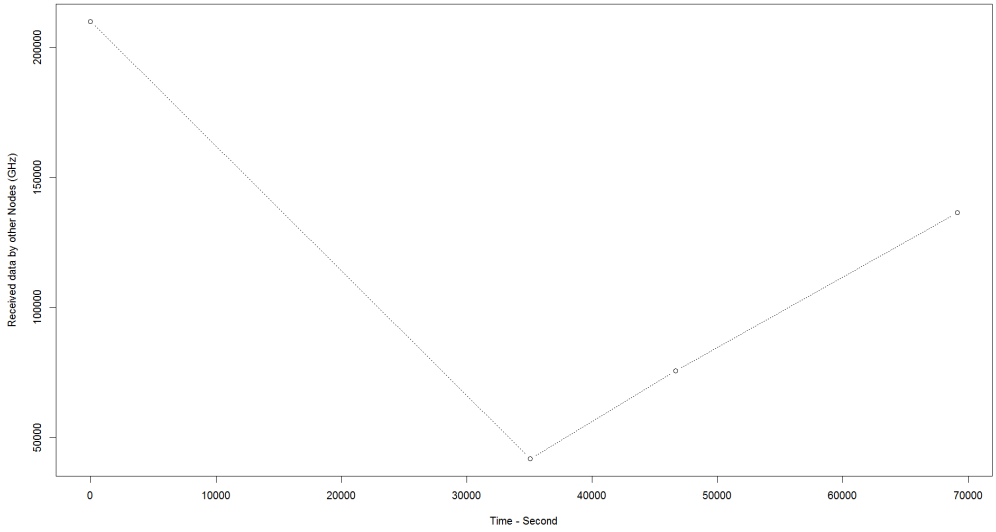

Figure H.17: How cloud management system views CPU resources with a *relative* threshold value 80, the updates points are plotted a circles.

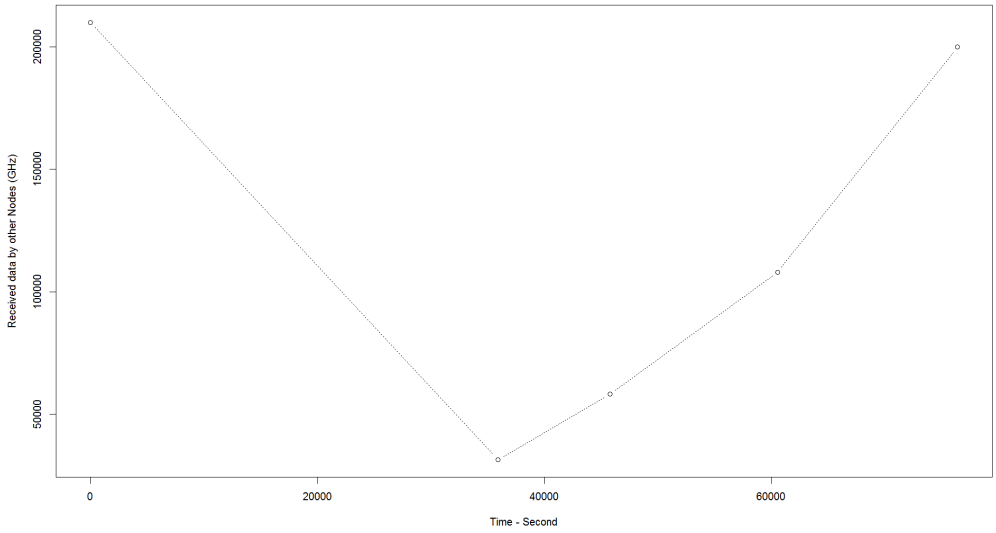

Figure H.18: How cloud management system views CPU resources with a *relative* threshold value 85, the updates points are plotted a circles.

H.2 Data center's RAM

Table H.2: Relative threshold update policy. Number of Cloud LSA updates based on changes in proposed data center's **RAM**. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
5	150	165	134	74	395	77.692
10	72	80	64	36	193	39.218
15	48	53	42	22	139	26.813
20	35	39	31	16	110	20.092
25	27	30	24	12	84	16.078
30	22	25	20	10	77	13.979
35	19	21	17	8	60	11.726
40	17	19	15	8	56	10.330
45	15	16	13	6	50	9.709
50	13	15	11	6	45	8.785
55	12	13	10	6	41	7.911
60	10	12	9	3	35	7.213
65	10	11	8	4	32	6.657
70	9	10	7	4	31	6.312
75	7	9	6	4	25	5.662
80	7	8	6	4	24	5.319
85	7	8	6	1	24	4.876
90	6	7	4	1	21	5.129
95	4	5	3	1	19	5.019

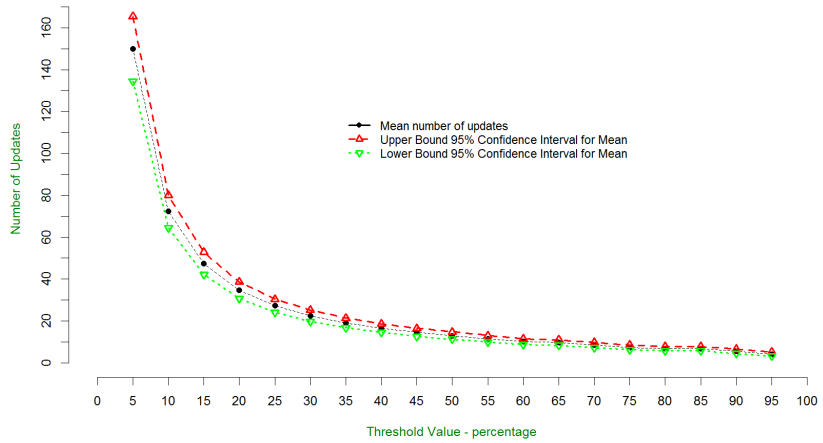

Figure H.19: Number of updates per different threshold values for *relative threshold-based* update policy based on changes of a data center's RAM capacity.

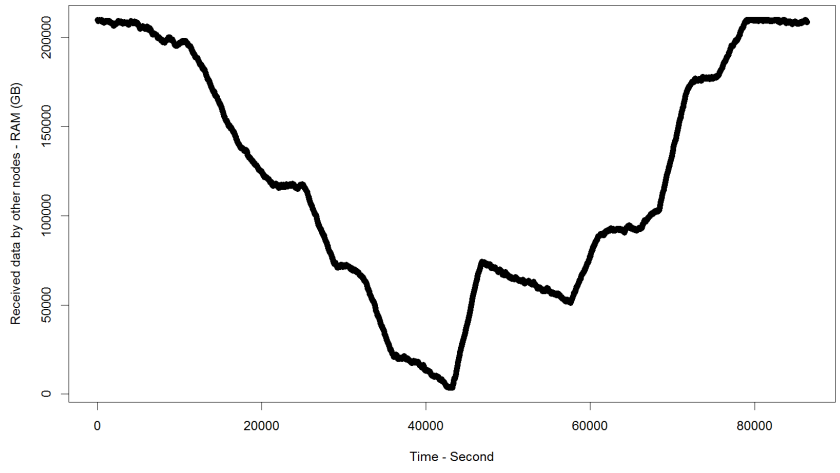

Figure H.20: Data center sample RAM (GHz) capacity per second

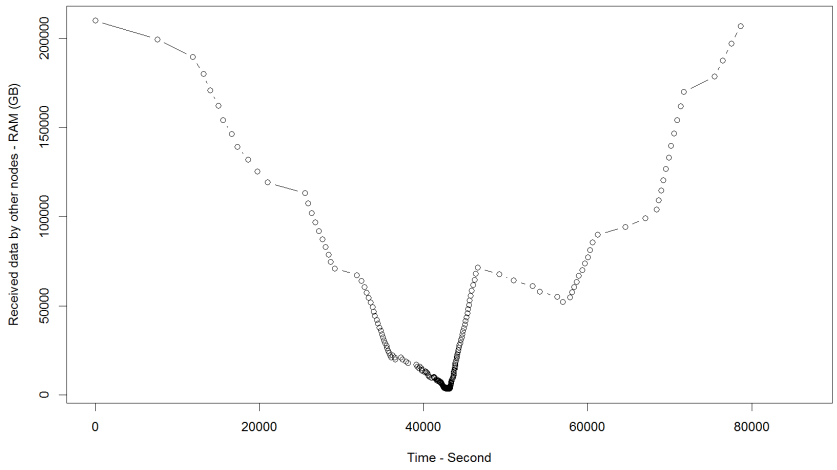

Figure H.21: How cloud management system views RAM resources with a *relative* threshold value 5, the updates points are plotted a circles.

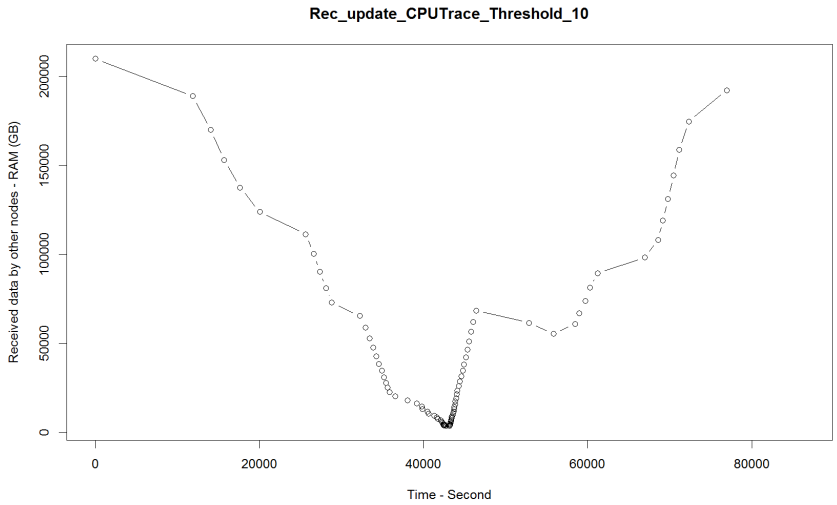

Figure H.22: How cloud management system views RAM resources with a *relative* threshold value 10, the updates points are plotted a circles.

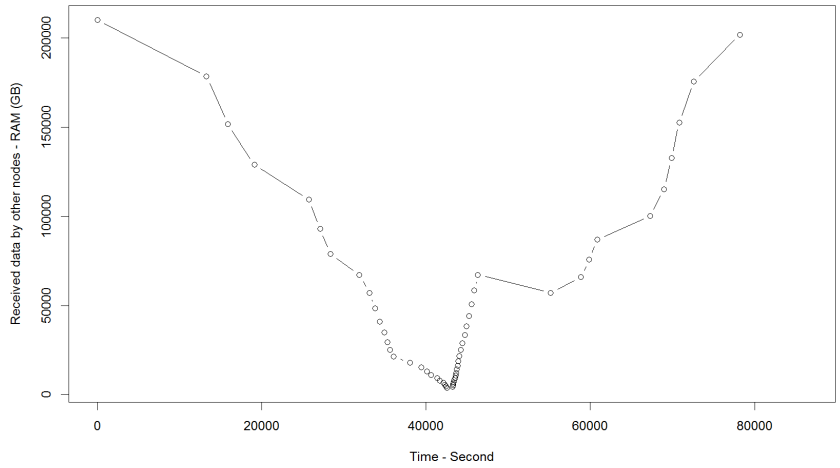

Figure H.23: How cloud management system views RAM resources with a *relative* threshold value 15, the updates points are plotted a circles.

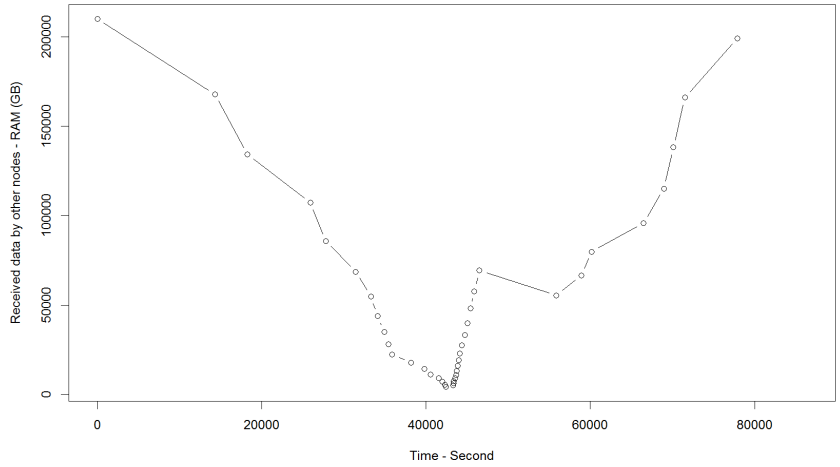

Figure H.24: How cloud management system views RAM resources with a *relative* threshold value 20, the updates points are plotted a circles.

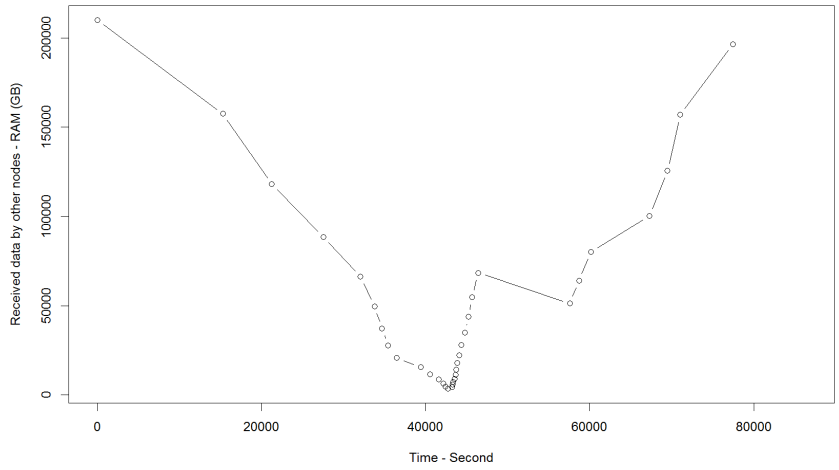

Figure H.25: How cloud management system views RAM resources with a *relative* threshold value 25, the updates points are plotted a circles.

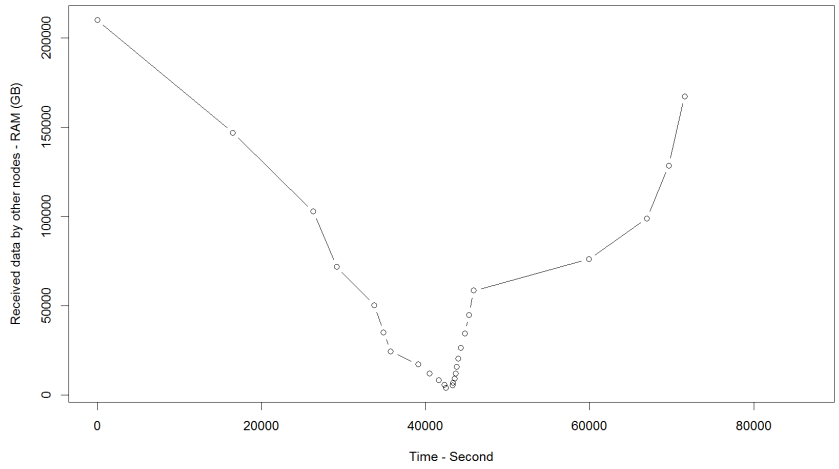

Figure H.26: How cloud management system views RAM resources with a *relative* threshold value 30, the updates points are plotted a circles.

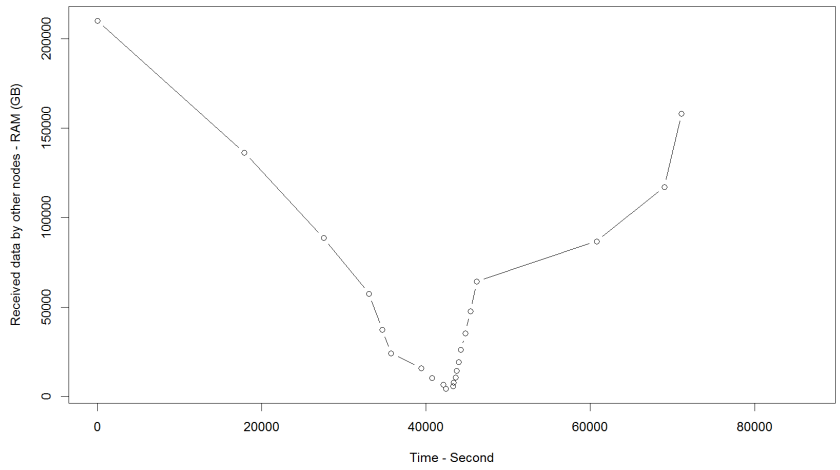

Figure H.27: How cloud management system views RAM resources with a *relative* threshold value 35, the updates points are plotted a circles.

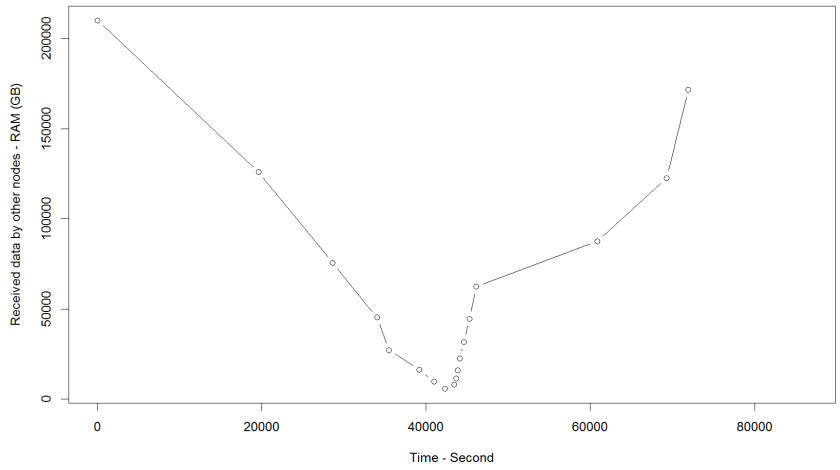

Figure H.28: How cloud management system views RAM resources with a *relative* threshold value 40, the updates points are plotted a circles.

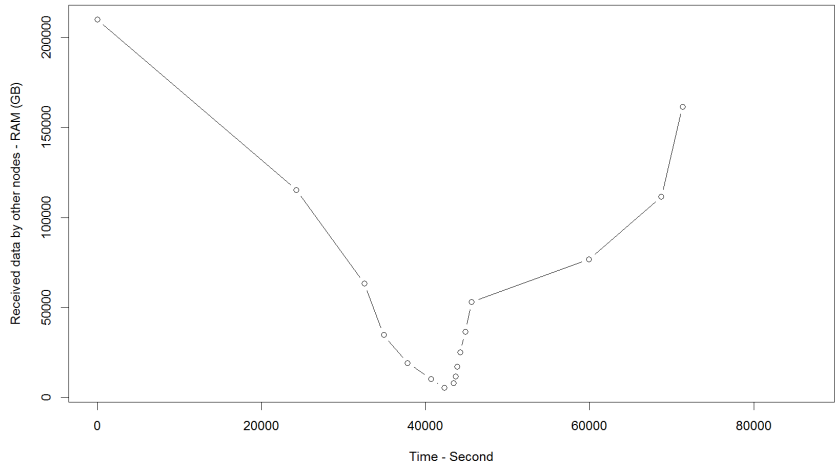

Figure H.29: How cloud management system views RAM resources with a *relative* threshold value 45, the updates points are plotted a circles.

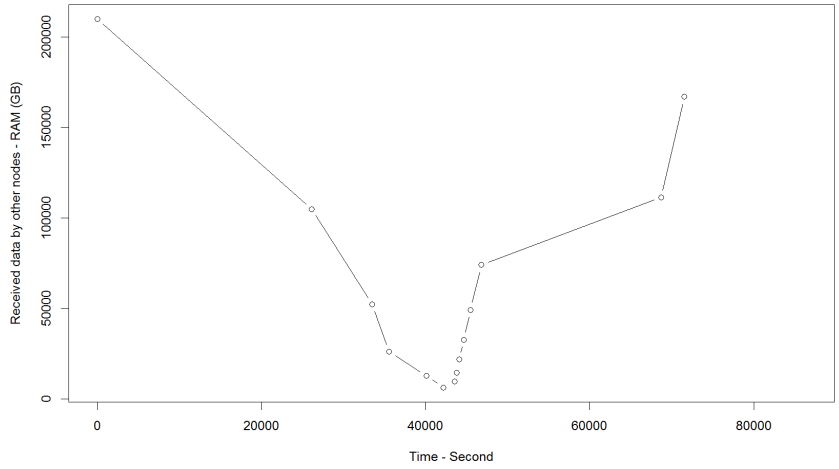

Figure H.30: How cloud management system views RAM resources with a *relative* threshold value 50, the updates points are plotted a circles.

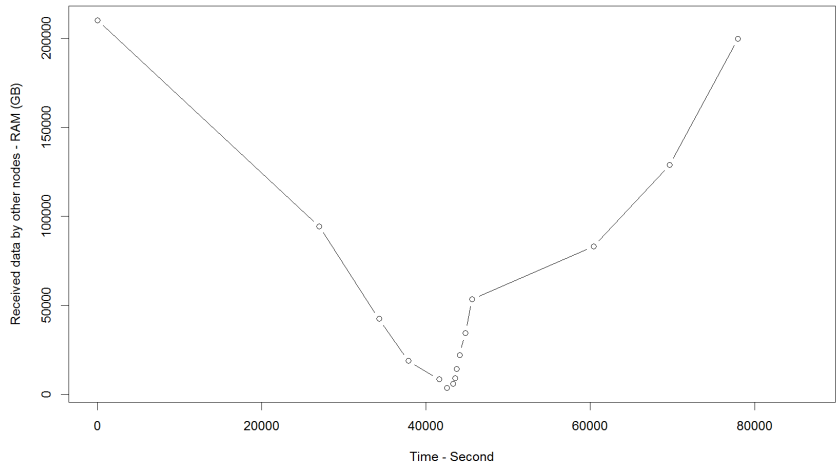

Figure H.31: How cloud management system views RAM resources with a *relative* threshold value 55, the updates points are plotted a circles.

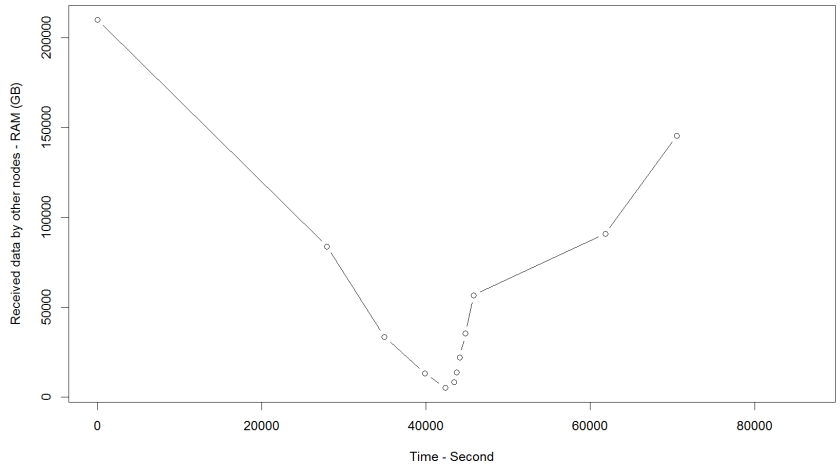

Figure H.32: How cloud management system views RAM resources with a *relative* threshold value 60, the updates points are plotted a circles.

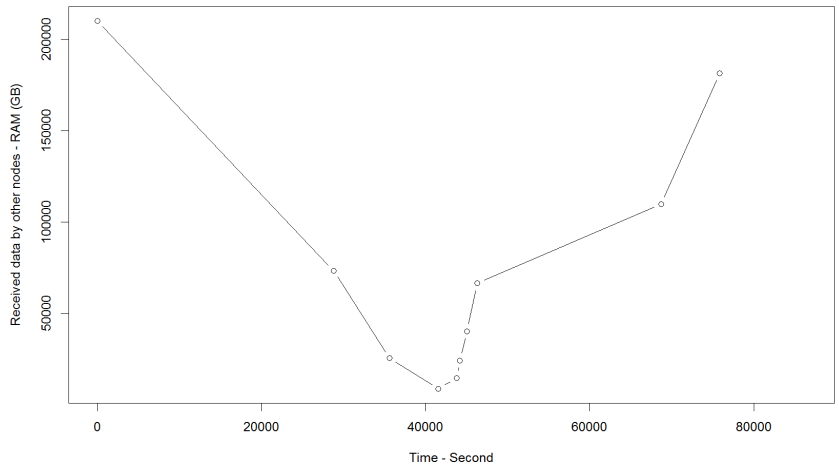

Figure H.33: How cloud management system views RAM resources with a *relative* threshold value 65, the updates points are plotted a circles.

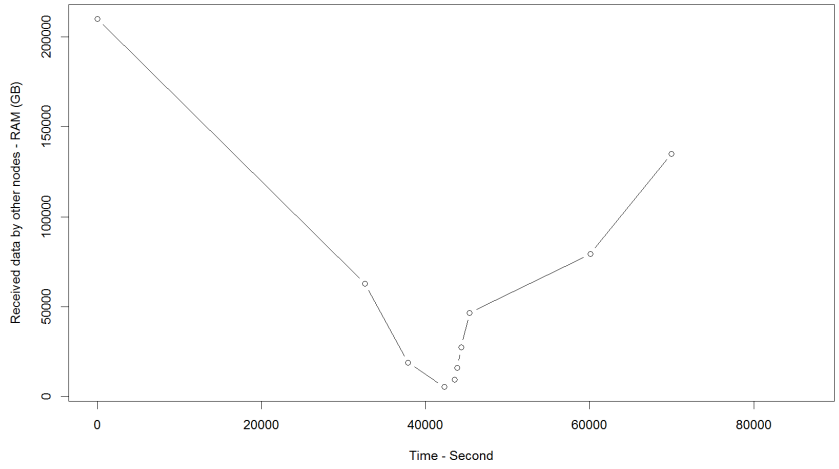

Figure H.34: How cloud management system views RAM resources with a *relative* threshold value 70, the updates points are plotted a circles.

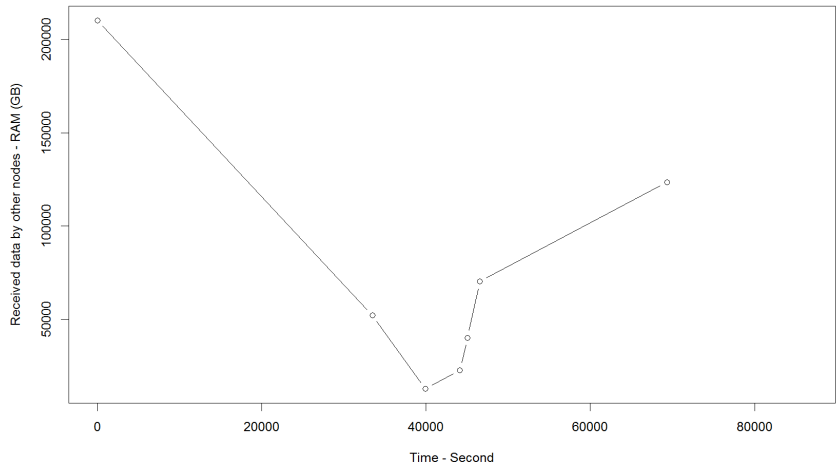

Figure H.35: How cloud management system views RAM resources with a *relative* threshold value 75, the updates points are plotted a circles.

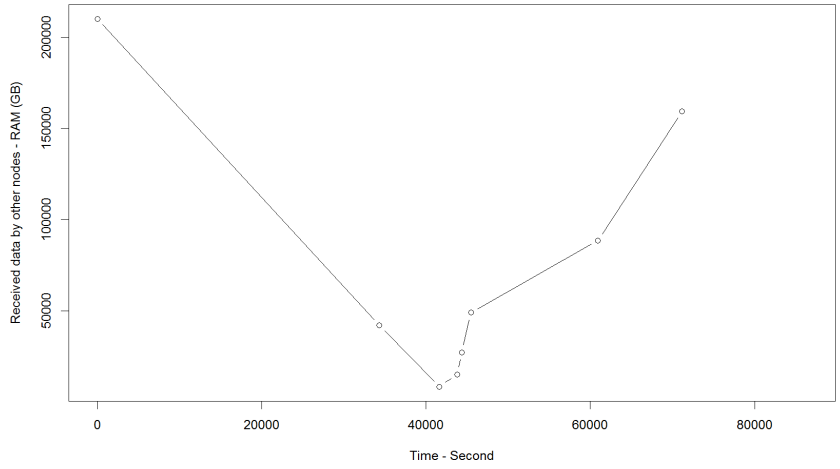

Figure H.36: How cloud management system views RAM resources with a *relative* threshold value 80, the updates points are plotted a circles.

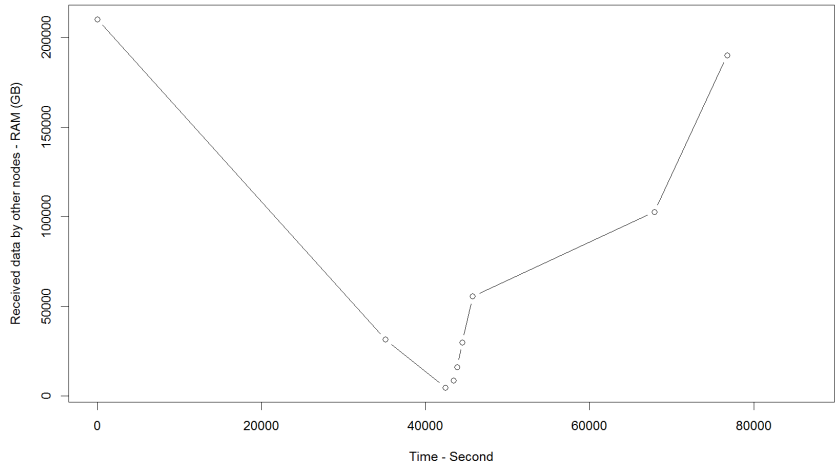

Figure H.37: How cloud management system views RAM resources with a *relative* threshold value 85, the updates points are plotted a circles.

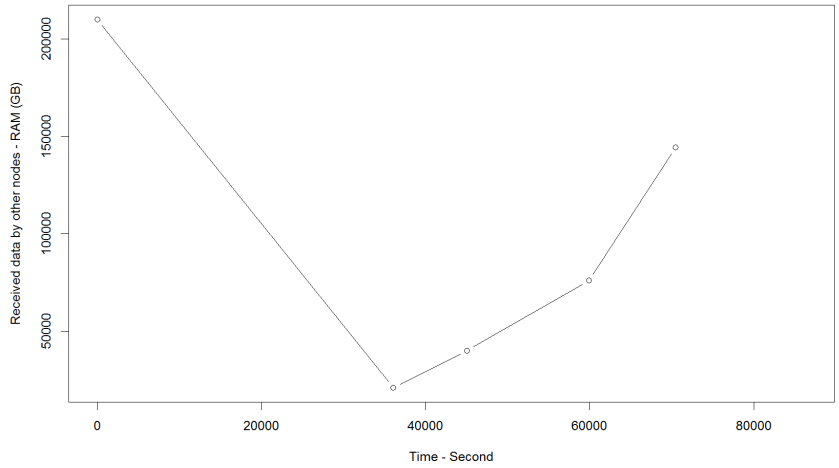

Figure H.38: How cloud management system views RAM resources with a *relative* threshold value 90, the updates points are plotted a circles.

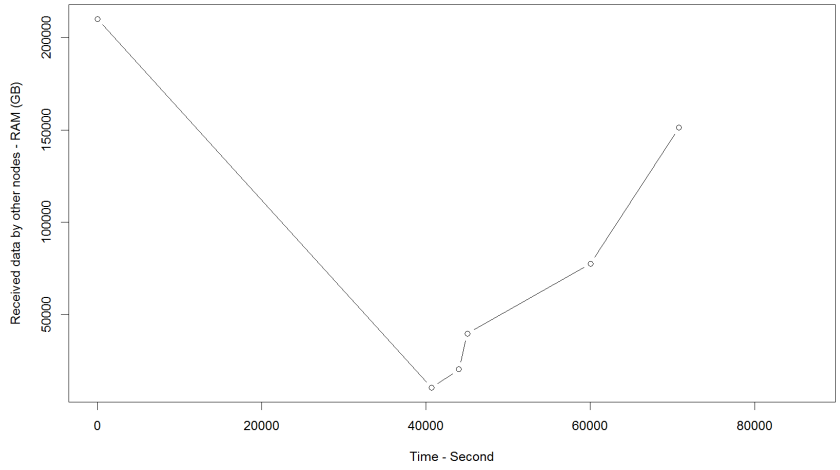

Figure H.39: How cloud management system views RAM resources with a *relative* threshold value 95, the updates points are plotted a circles.

H.3 Data center's storage

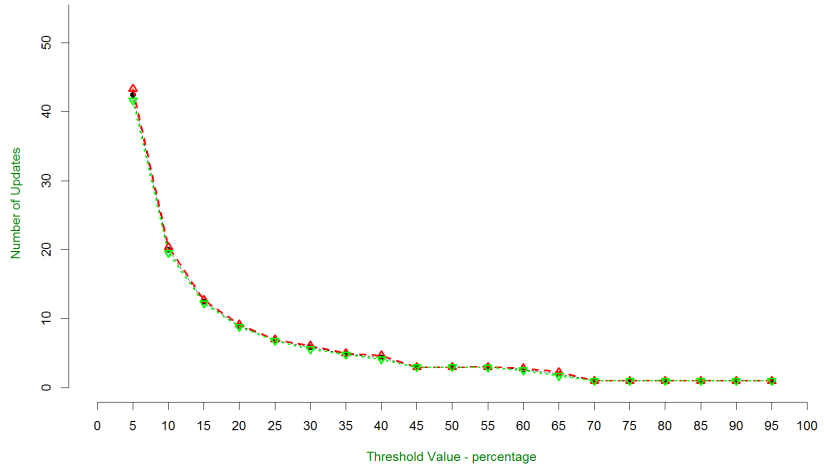

Figure H.40: Number of updates per different threshold values for *relative threshold-based* update policy based on changes of a data center's storage capacity.

Table H.3: Relative threshold update policy. Number of Cloud LSA updates based on changes in proposed data center's **storage**. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
5	42	43	42	33	52	4.239
10	20	20	20	15	24	2.229
15	12	13	12	9	16	1.235
20	9	9	9	7	12	0.953
25	7	7	7	5	7	0.438
30	6	6	6	5	8	1.343
35	5	5	5	3	5	0.438
40	4	5	4	3	6	1.477
45	3	3	3	3	3	0.000
50	3	3	3	3	3	0.000
55	3	3	3	1	3	0.200
60	3	3	3	1	3	0.736
65	2	2	2	1	4	1.417
70	1	1	1	1	1	0.000
75	1	1	1	1	1	0.000
80	1	1	1	1	1	0.000
85	1	1	1	1	1	0.000
90	1	1	1	1	1	0.000
95	1	1	1	1	1	0.000

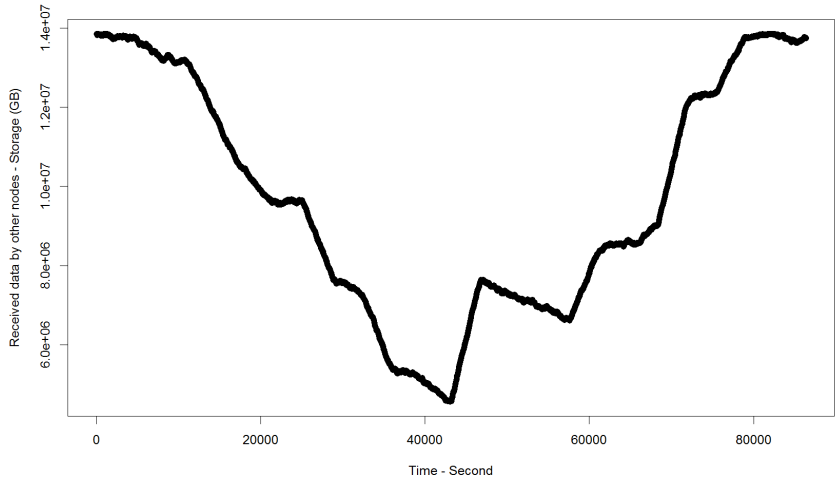

Figure H.41: Data center sample storage (GB) capacity per second.

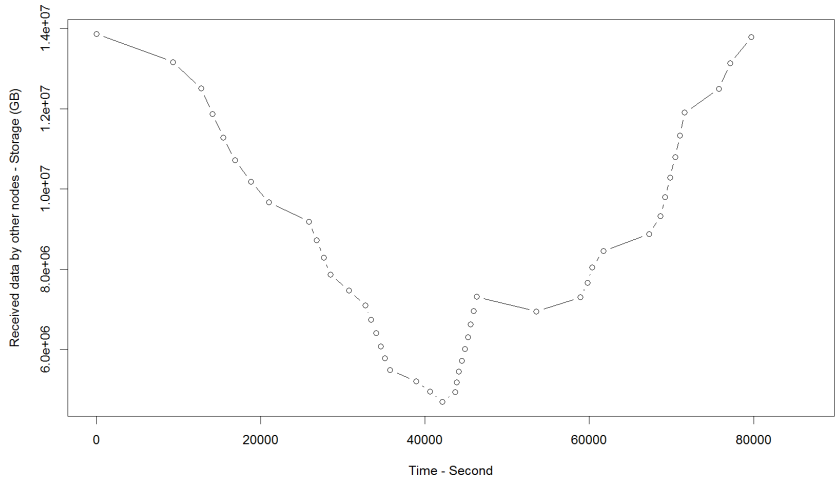

Figure H.42: How cloud management system views storage resources with a *relative* threshold value 5, the updates points are plotted a circles.

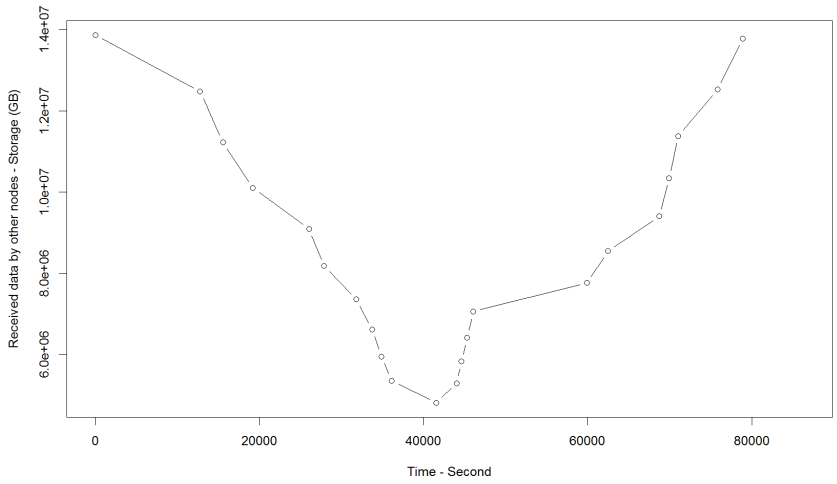

Figure H.43: How cloud management system views storage resources with a *relative* threshold value 10, the updates points are plotted a circles.

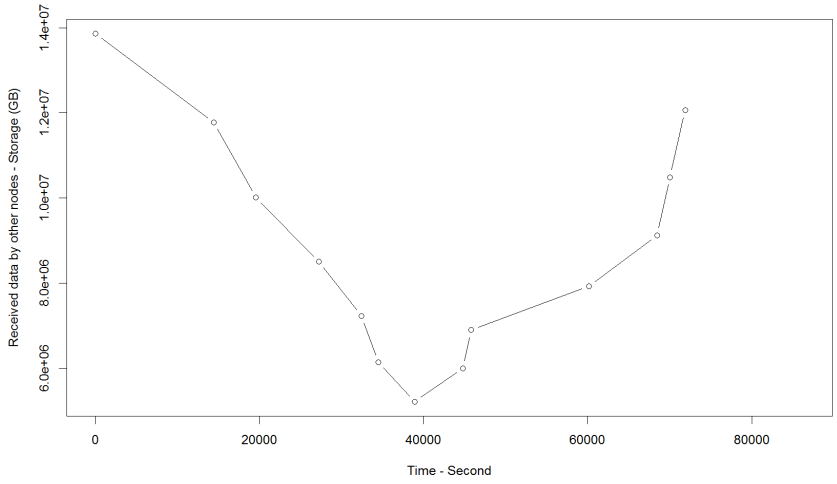

Figure H.44: How cloud management system views storage resources with a *relative* threshold value 15, the updates points are plotted a circles.

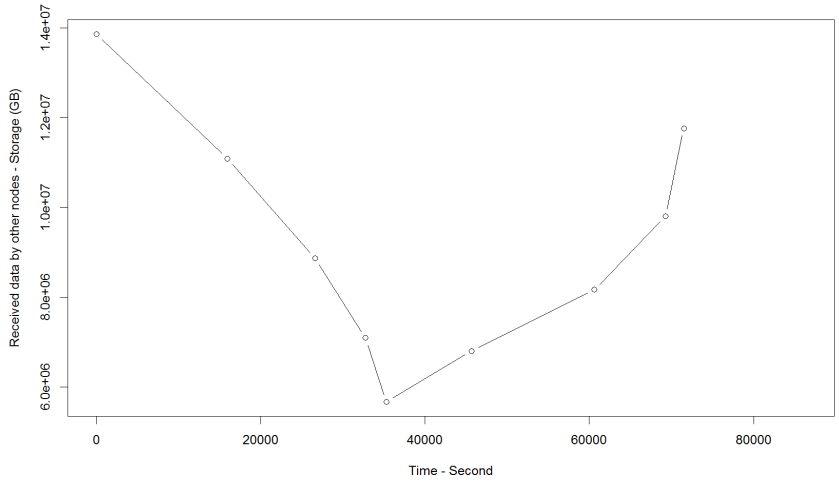

Figure H.45: How cloud management system views storage resources with a *relative* threshold value 20, the updates points are plotted a circles.

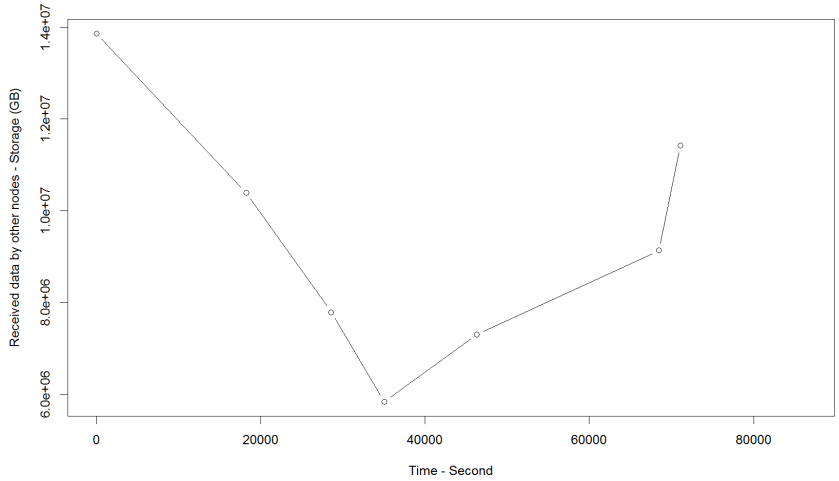

Figure H.46: How cloud management system views storage resources with a *relative* threshold value 25, the updates points are plotted a circles.

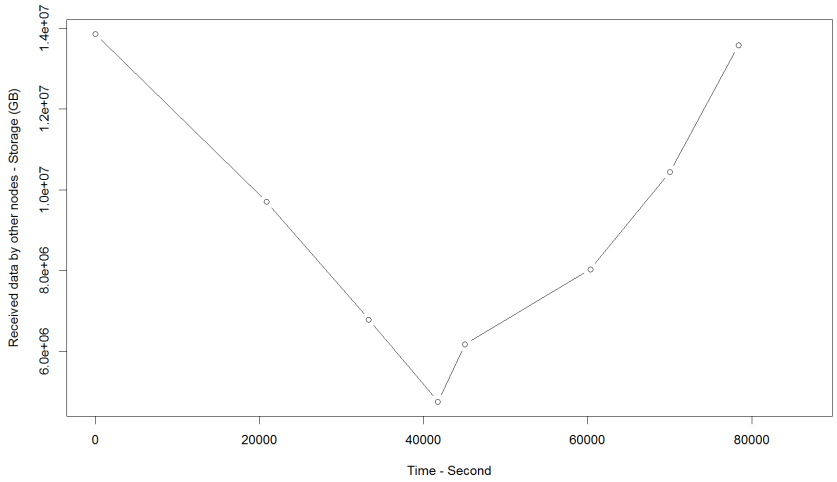

Figure H.47: How cloud management system views storage resources with a *relative* threshold value 30, the updates points are plotted a circles.

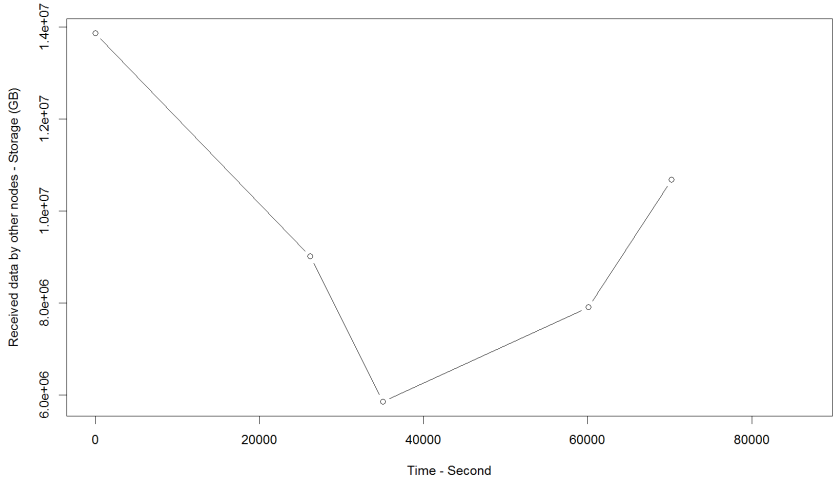

Figure H.48: How cloud management system views storage resources with a *relative* threshold value 35, the updates points are plotted a circles.

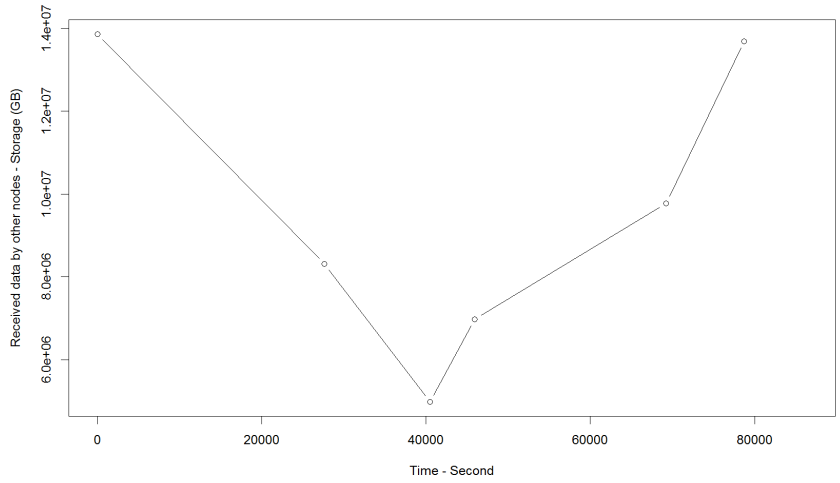

Figure H.49: How cloud management system views storage resources with a *relative* threshold value 40, the updates points are plotted a circles.

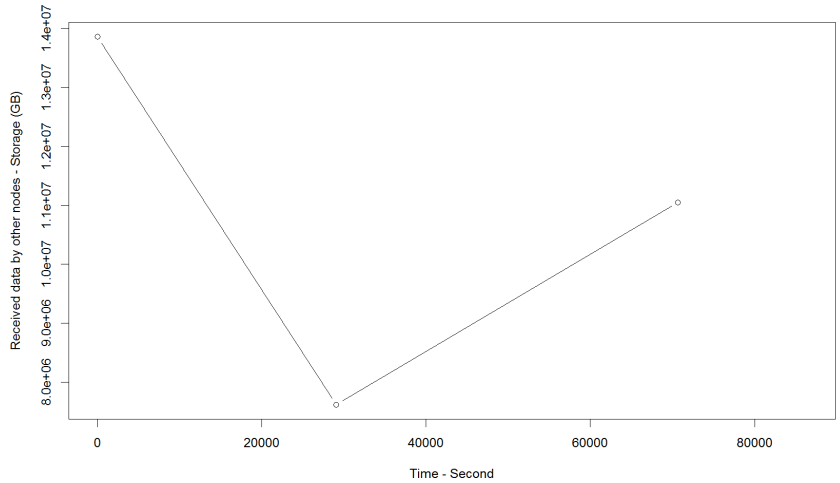

Figure H.50: How cloud management system views storage resources with a *relative* threshold value 45, the updates points are plotted a circles.

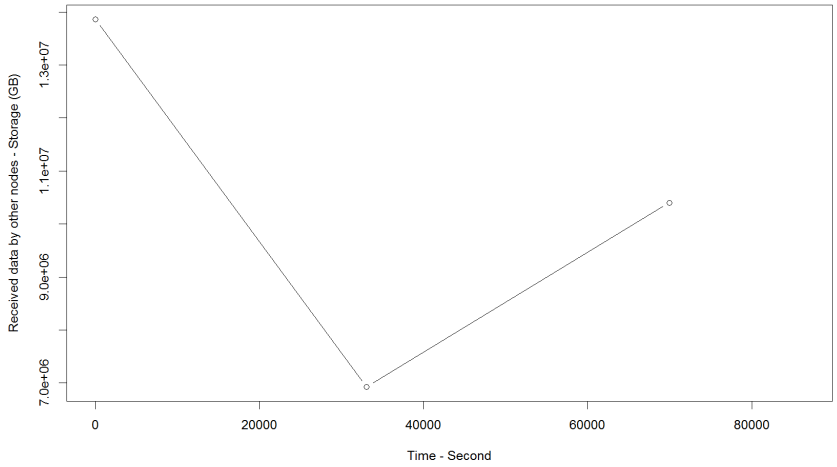

Figure H.51: How cloud management system views storage resources with a *relative* threshold value 50, the updates points are plotted a circles.

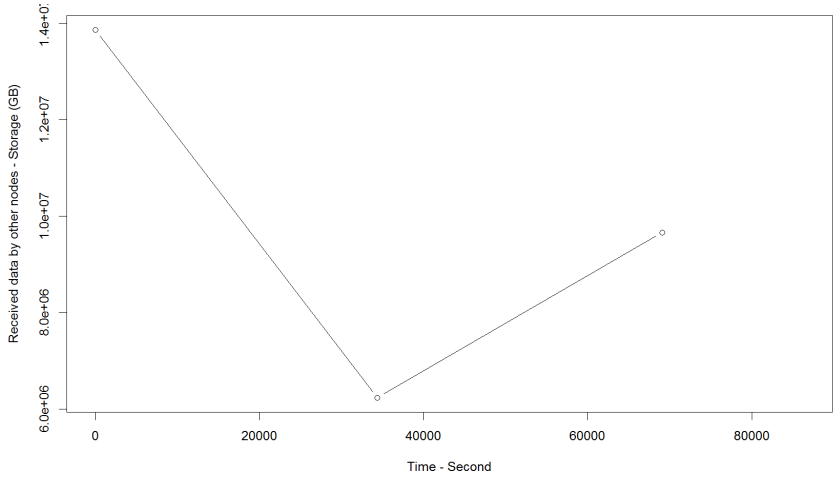

Figure H.52: How cloud management system views storage resources with a *relative* threshold value 55, the updates points are plotted a circles.

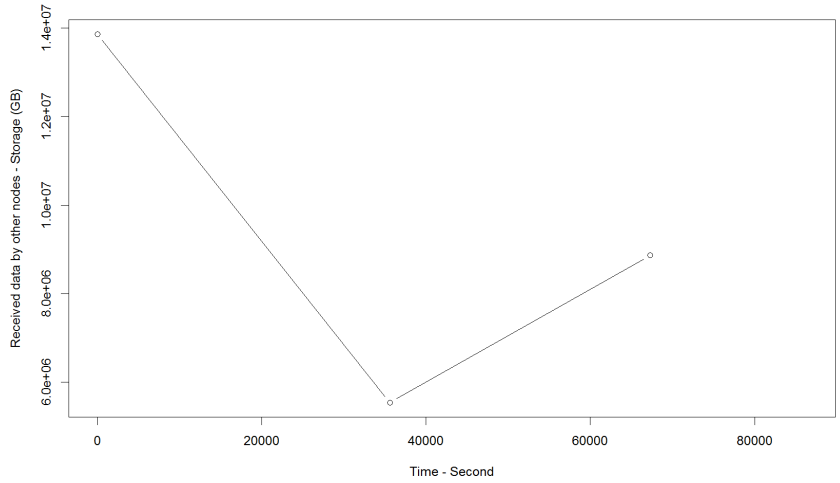

Figure H.53: How cloud management system views storage resources with a *relative* threshold value 60, the updates points are plotted a circles.

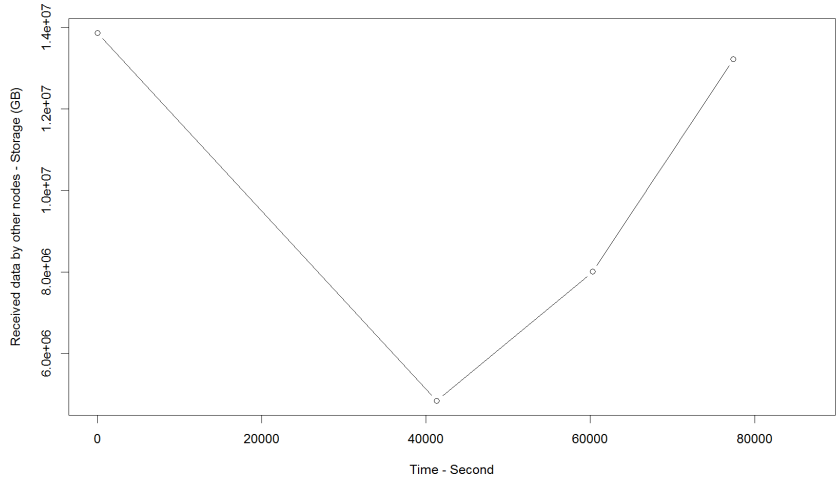

Figure H.54: How cloud management system views storage resources with a *relative* threshold value 65, the updates points are plotted a circles.

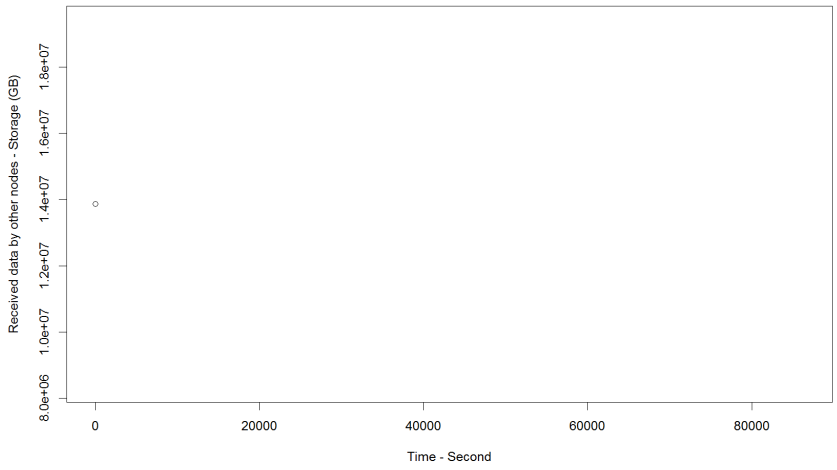

Figure H.55: How cloud management system views storage resources with a *relative* threshold value 70, the updates points are plotted a circles.

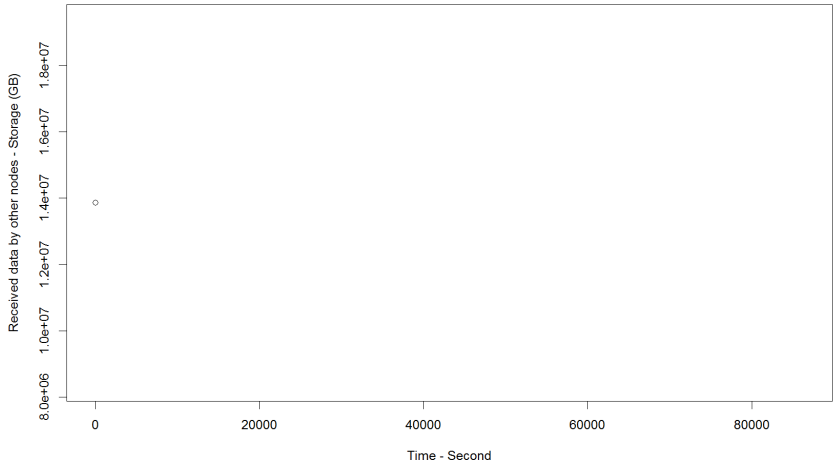

Figure H.56: How cloud management system views storage resources with a *relative* threshold value 75, the updates points are plotted a circles.

Appendix I

Relative threshold update policy experiments (2)

This Appendix provides complete set of results when a relative threshold update policy with different threshold values (e.g., 5, 10, 15, 20, and etc) was applied on a proposed simulated data center's resources (see section 5.4.2). It is worth mentioning that in this Appendix we assumes the update generates when changes in each of data center's resources (i.e., CPU, RAM, and storage) exceed the relative threshold value, as discussed in algorithm 1 on page 90. Table I.1 and Figure I.1 show the relation between the relative threshold value and number of updates.

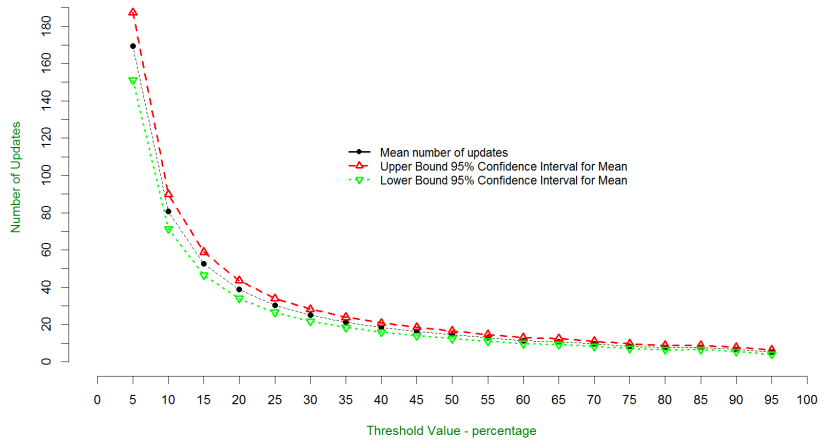

Figure I.1: Number of updates per different threshold values for *relative threshold-based* update policy based on changes of a data center's capacity.

Table I.1: Relative threshold update policy. Number of Cloud LSA updates based on changes in proposed data center's resource. In this Table "Thr" refers to a *"Threshold value"*, "LB" refers to a *"95% confidence interval Lower Bound"*, "UP" refers to a *"95% confidence interval Upper Bound"*, and "Std." refers to a *"Standard Deviation"*. The updates are trigger as discussed in algorithm 1 on page 90.

Thr	**Number of updates**					
(%)	**Mean**	**UB**	**LB**	**Min**	**Max**	**Std.**
5	169	187	151	76	539	91.700
10	81	90	71	36	281	46.746
15	53	59	46	22	173	31.364
20	39	43	34	16	145	23.975
25	30	34	26	12	105	18.917
30	25	28	22	10	92	16.425
35	21	24	18	8	80	13.943
40	18	21	16	8	72	12.329
45	16	18	14	6	64	11.414
50	14	16	12	6	54	10.241
55	13	15	11	6	48	9.182
60	11	13	10	3	44	8.517
65	11	12	9	4	40	7.747
70	10	11	8	4	36	7.292
75	8	10	7	4	29	6.569
80	8	9	6	4	30	6.338
85	8	9	6	1	29	5.733
90	7	8	5	1	26	5.859
95	5	6	4	1	24	5.836

Additionally, section I.1, section I.2, and section I.3 show how the receiver node(s) views data center's resources (i.e., CPU, RAM, and storage) with different relative threshold values when an algorithm 1 is used for triggering a new Cloud LSA update. The points that shown in circle in these graphs depict the time that the link-state update is sent out into the network. These Figures provide an overview to help to understand how other node(s) in the network views the data center's resources based on each threshold value.

I.1 Data center's CPU

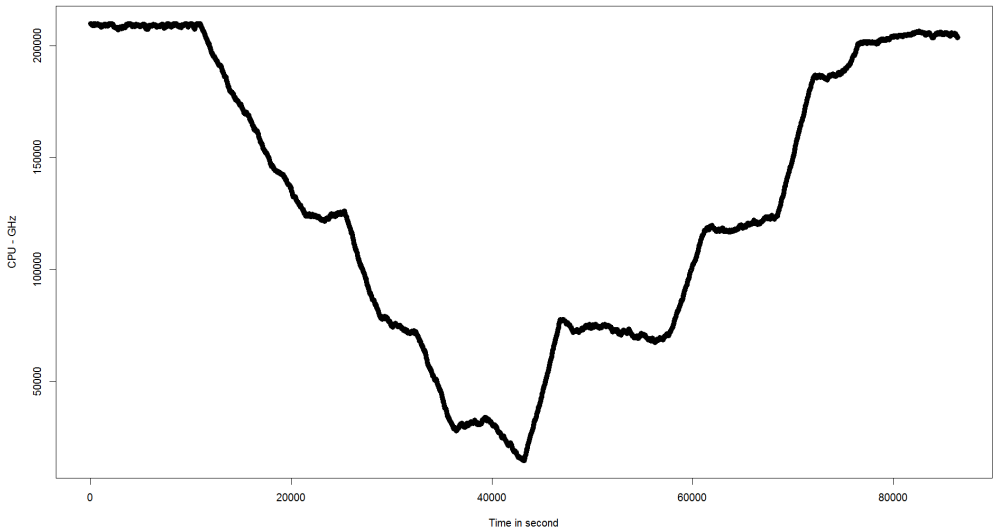

Figure I.2: Data center sample CPU (GHz) capacity per second

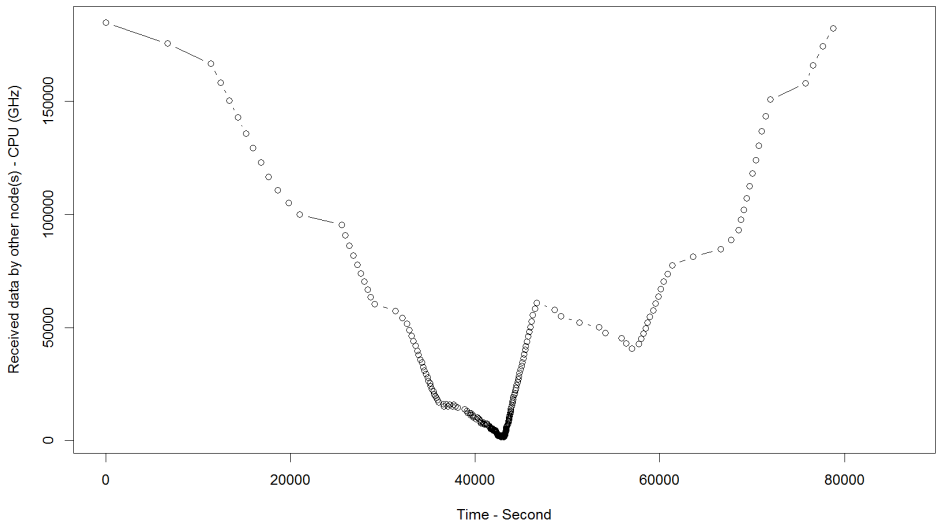

Figure I.3: How a cloud management system views CPU resources with a *relative* threshold value 5, the updates points are plotted a circles.

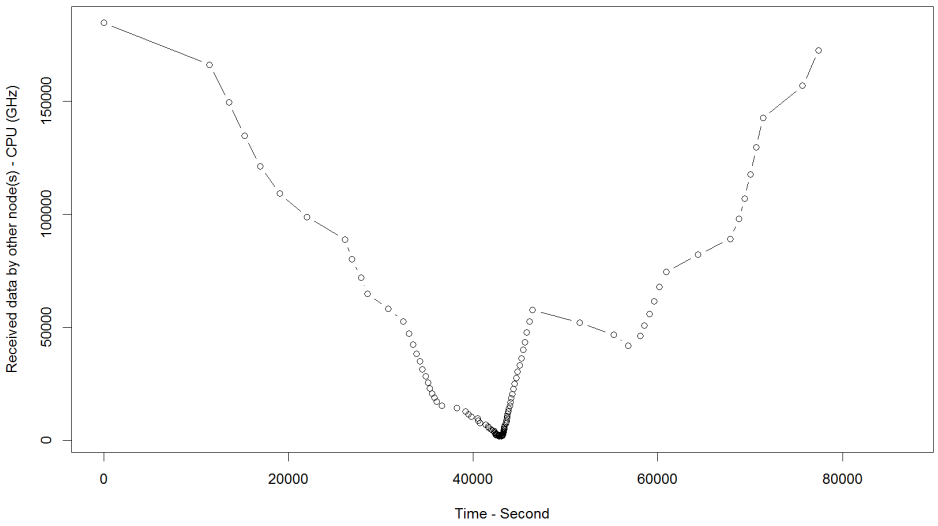

Figure I.4: How a cloud management system views CPU resources with a *relative* threshold value 10, the updates points are plotted a circles.

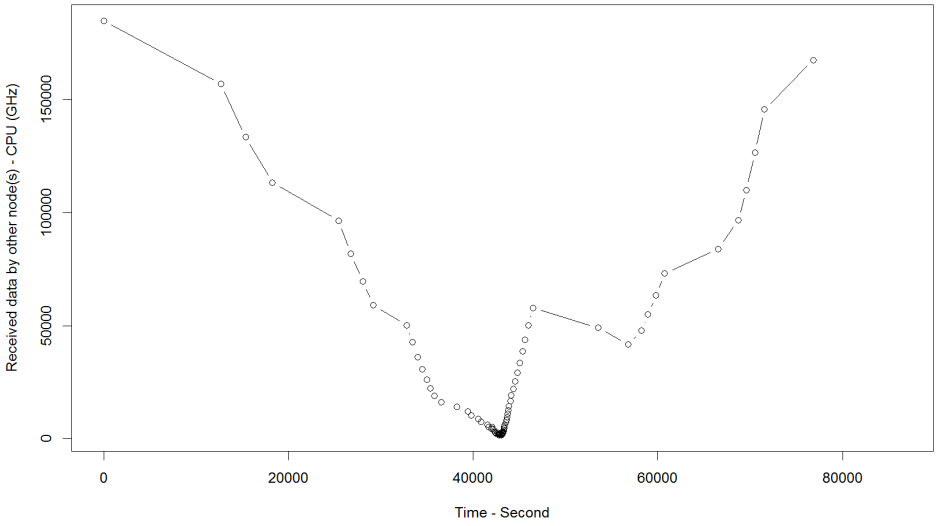

Figure I.5: How a cloud management system views CPU resources with a *relative* threshold value 15, the updates points are plotted a circles.

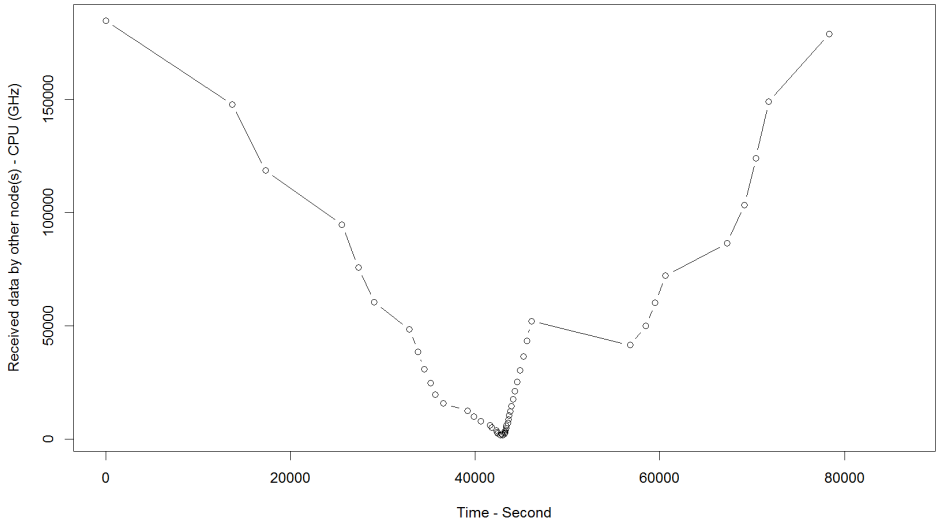

Figure I.6: How a cloud management system views CPU resources with a *relative* threshold value 20, the updates points are plotted a circles.

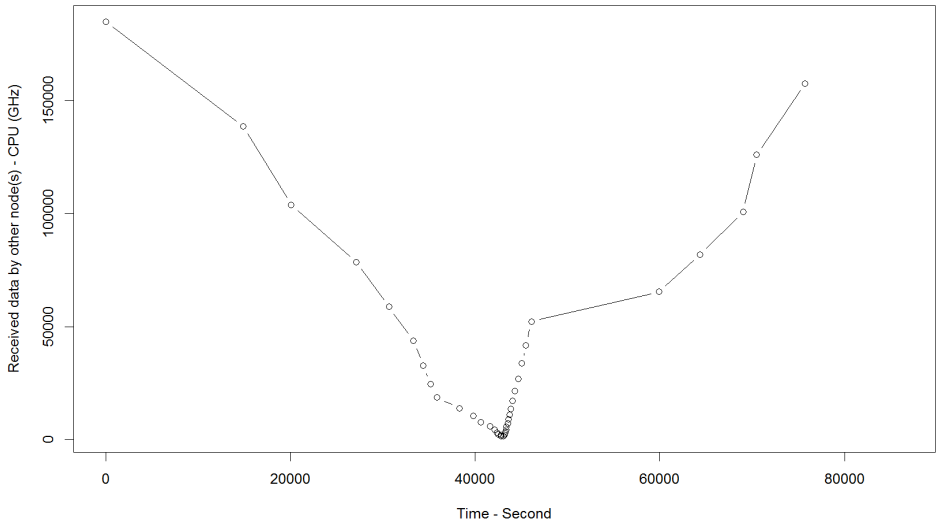

Figure I.7: How a cloud management system views CPU resources with a *relative* threshold value 25, the updates points are plotted a circles.

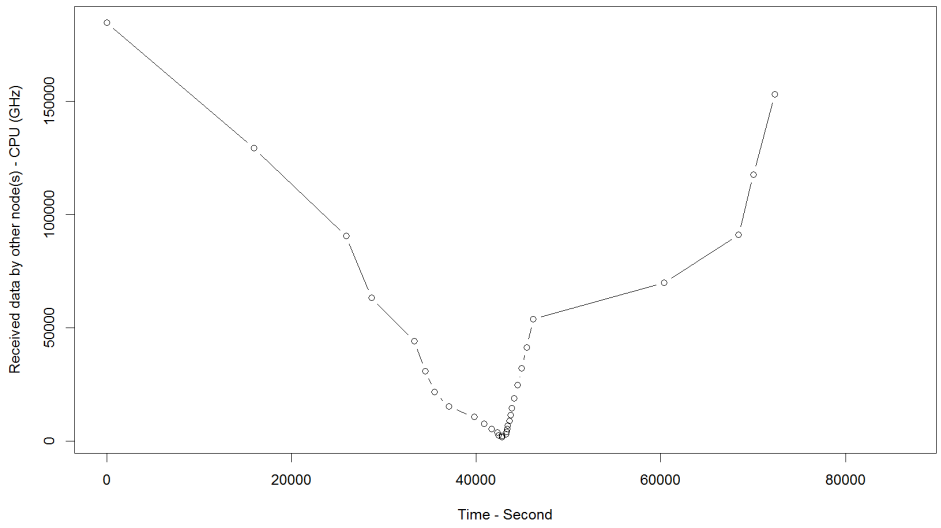

Figure I.8: How a cloud management system views CPU resources with a *relative* threshold value 30, the updates points are plotted a circles.

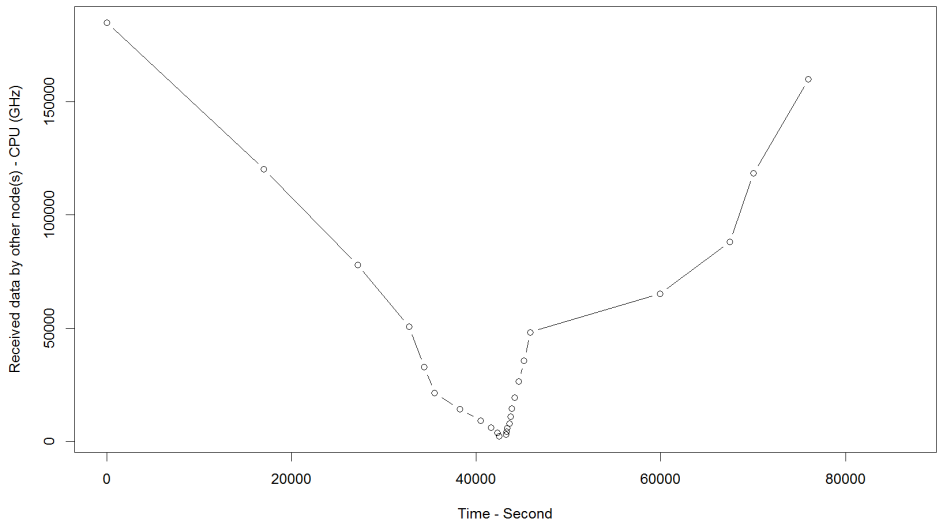

Figure I.9: How a cloud management system views CPU resources with a *relative* threshold value 35, the updates points are plotted a circles.

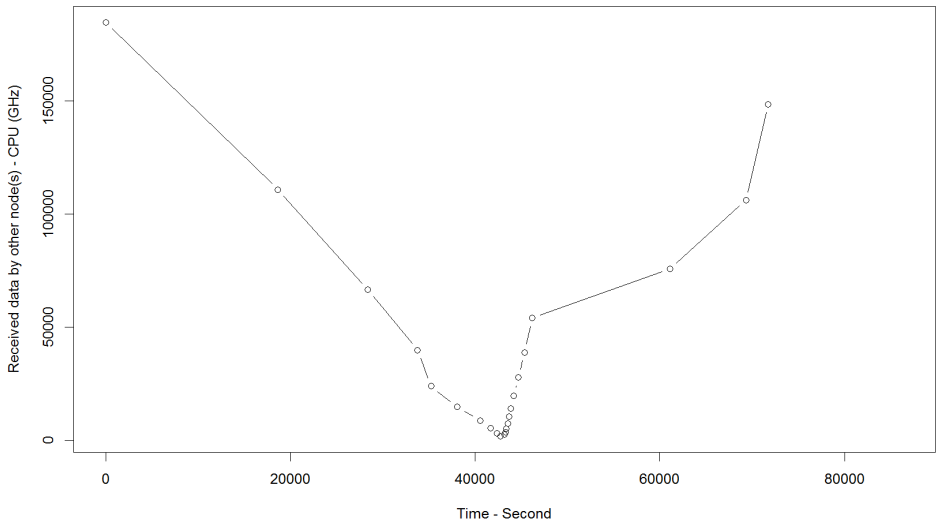

Figure I.10: How a cloud management system views CPU resources with a *relative* threshold value 40, the updates points are plotted a circles.

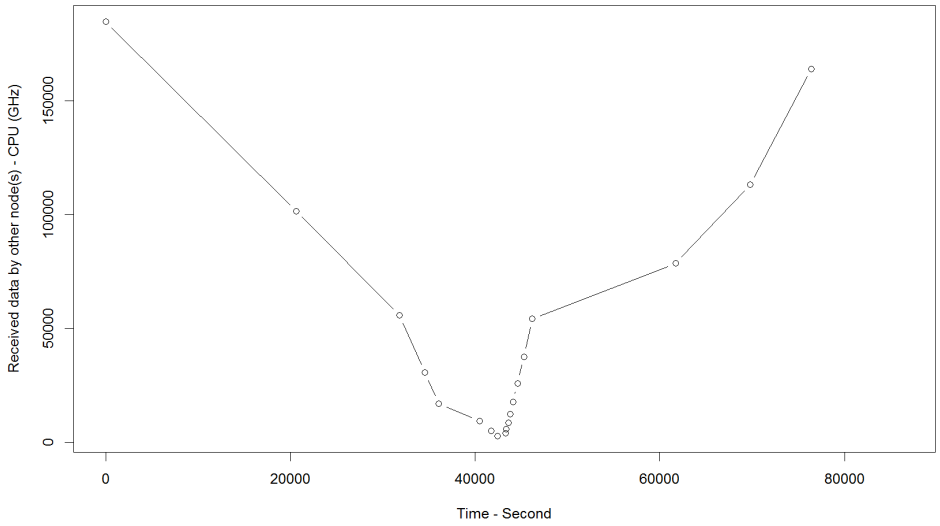

Figure I.11: How a cloud management system views CPU resources with a *relative* threshold value 45, the updates points are plotted a circles.

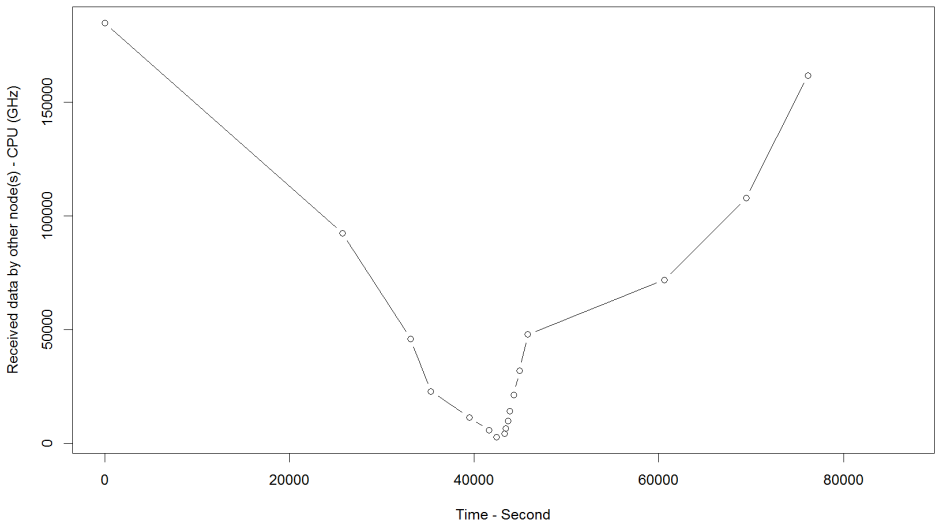

Figure I.12: How a cloud management system views CPU resources with a *relative* threshold value 50, the updates points are plotted a circles.

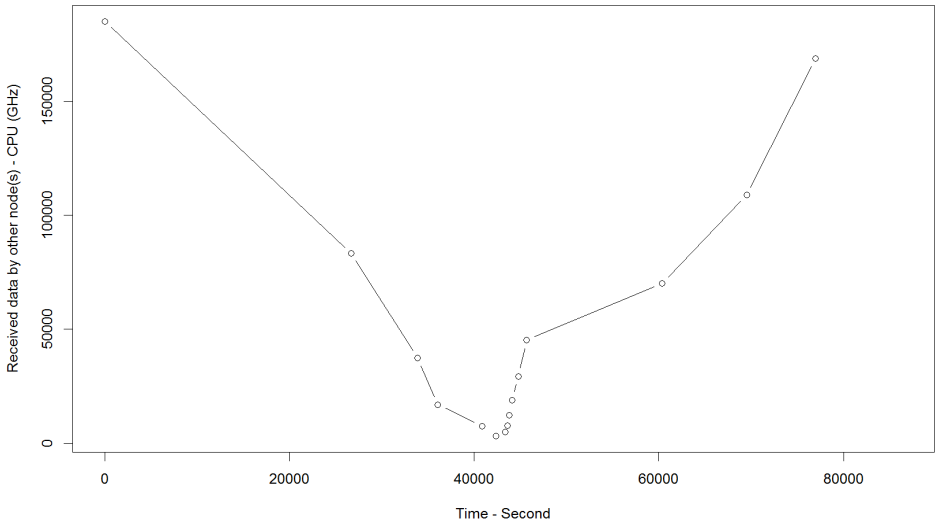

Figure I.13: How a cloud management system views CPU resources with a *relative* threshold value 55, the updates points are plotted a circles.

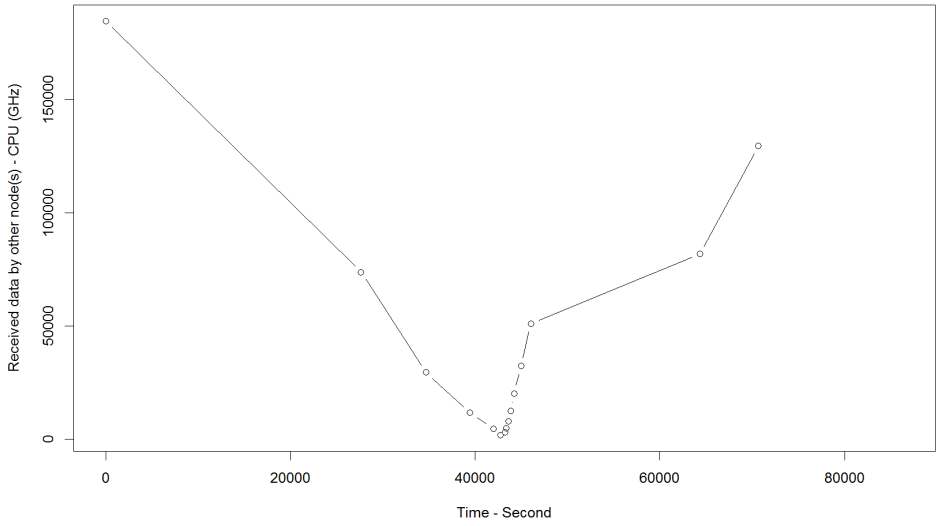

Figure I.14: How a cloud management system views CPU resources with a *relative* threshold value 60, the updates points are plotted a circles.

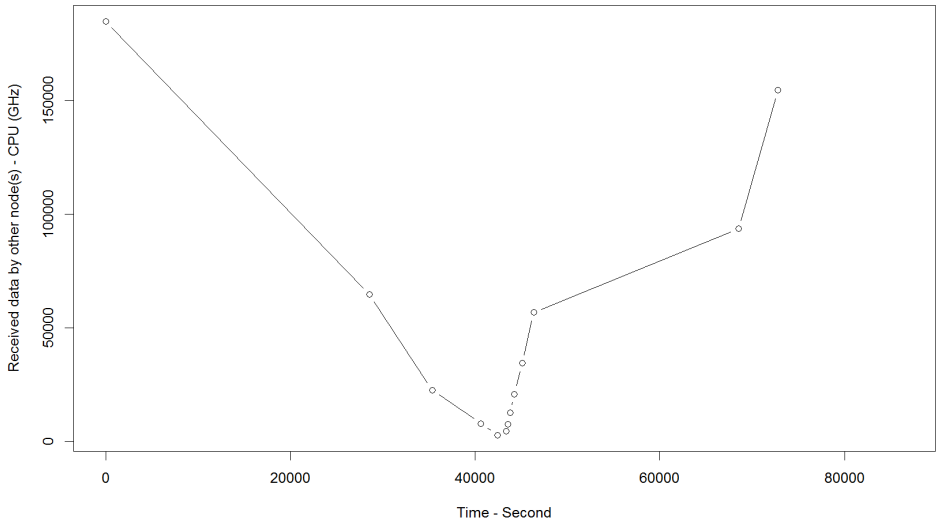

Figure I.15: How a cloud management system views CPU resources with a *relative* threshold value 65, the updates points are plotted a circles.

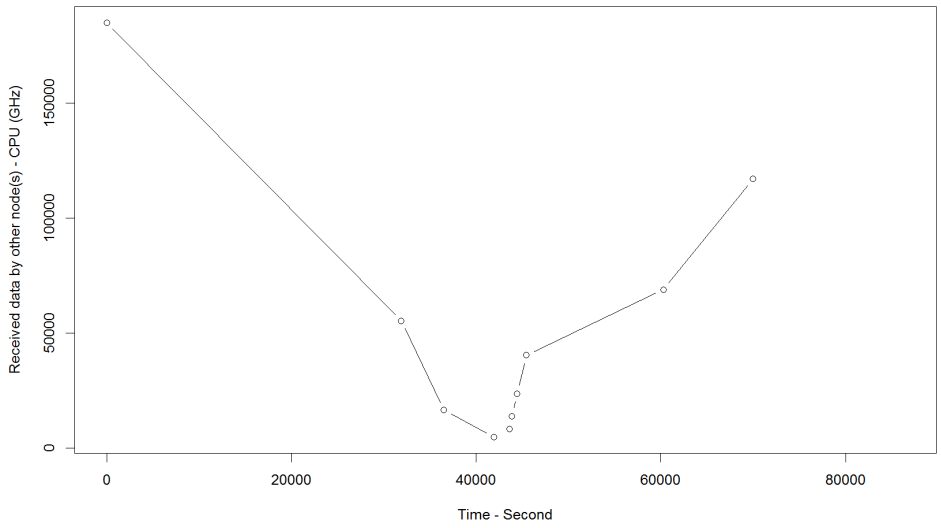

Figure I.16: How a cloud management system views CPU resources with a *relative* threshold value 70, the updates points are plotted a circles.

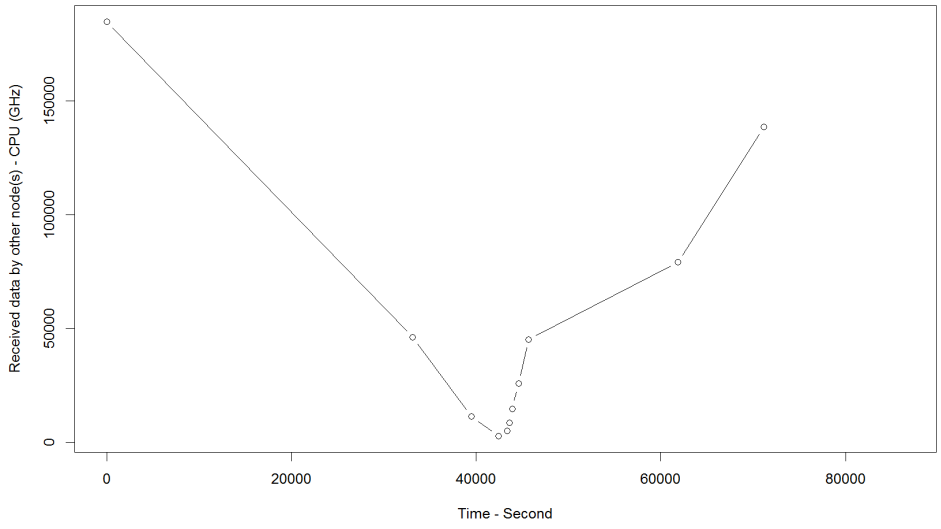

Figure I.17: How a cloud management system views CPU resources with a *relative* threshold value 75, the updates points are plotted a circles.

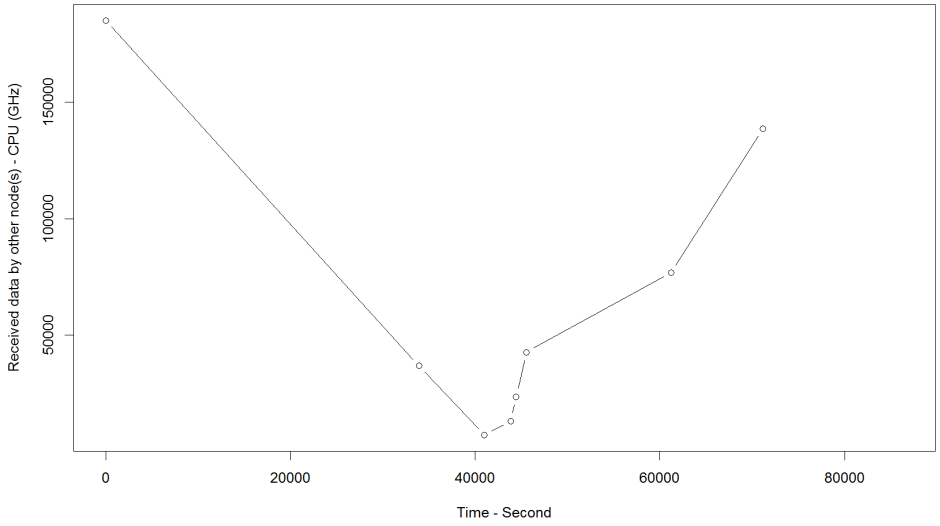

Figure I.18: How a cloud management system views CPU resources with a *relative* threshold value 80, the updates points are plotted a circles.

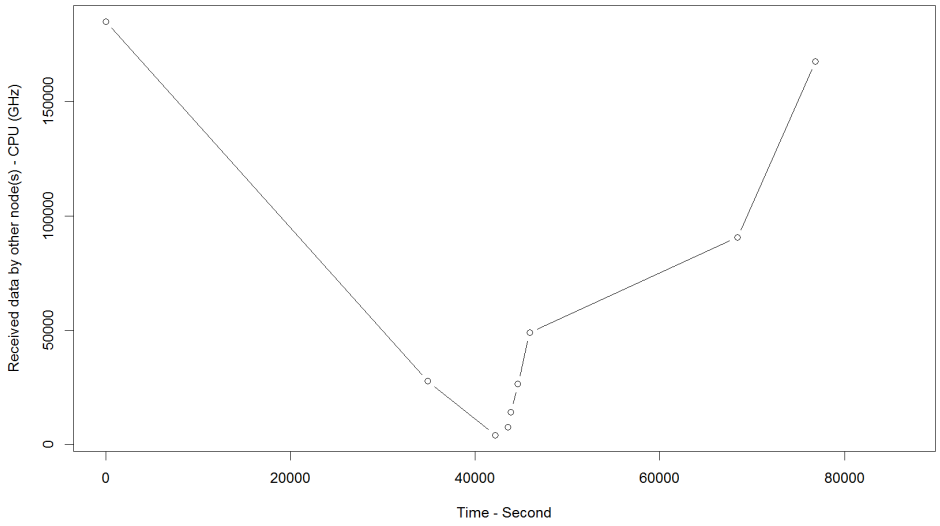

Figure I.19: How a cloud management system views CPU resources with a *relative* threshold value 85, the updates points are plotted a circles.

I.2 Data center's RAM

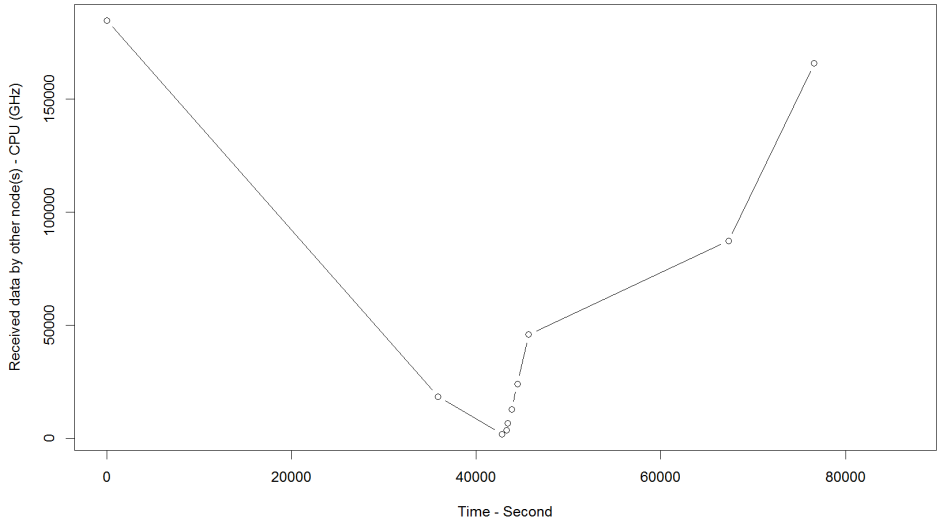

Figure I.20: How a cloud management system views CPU resources with a *relative* threshold value 90, the updates points are plotted a circles.

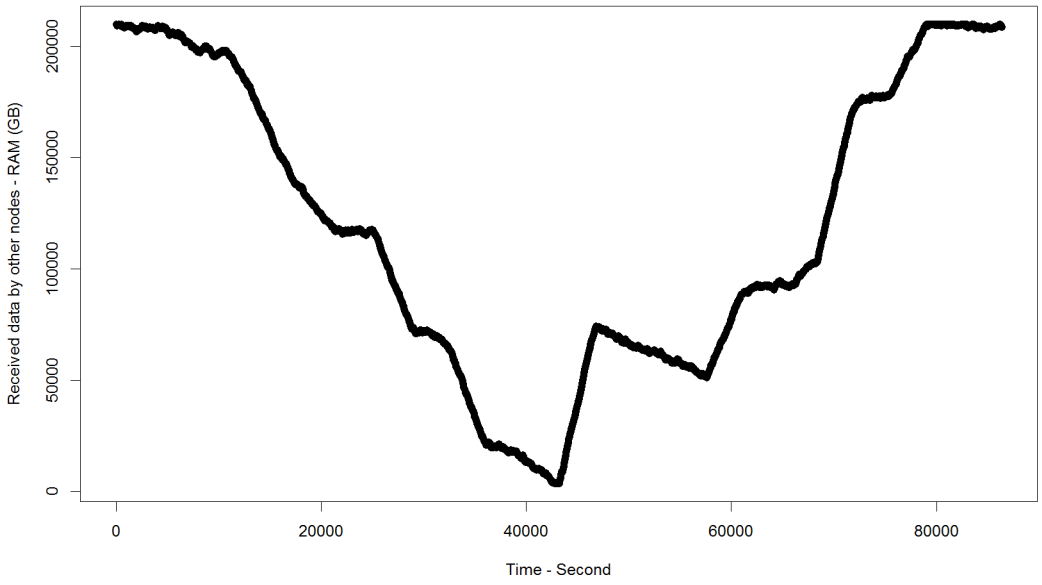

Figure I.21: Data center sample RAM (GB) capacity per second

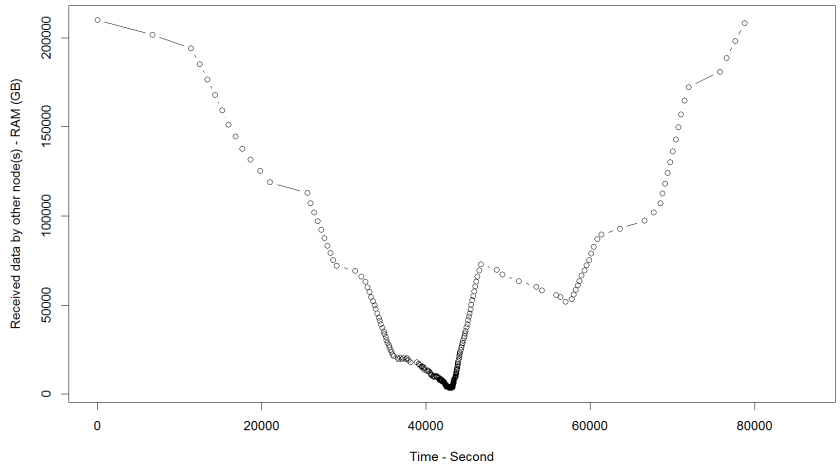

Figure I.22: How a cloud management system views RAM resources with a *relative* threshold value 5, the updates points are plotted a circles.

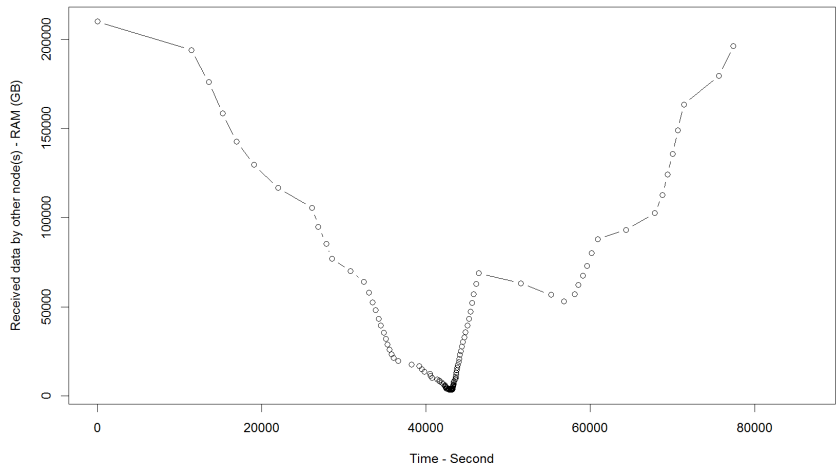

Figure I.23: How a cloud management system views RAM resources with a *relative* threshold value 10, the updates points are plotted a circles.

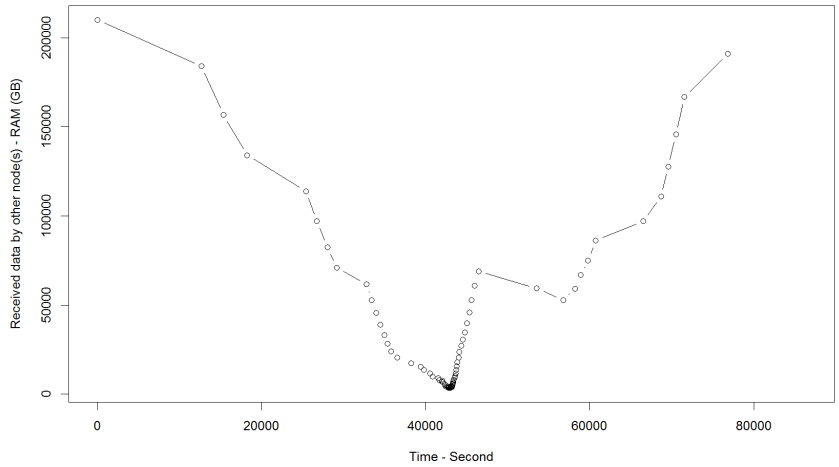

Figure I.24: How a cloud management system views RAM resources with a *relative* threshold value 15, the updates points are plotted a circles.

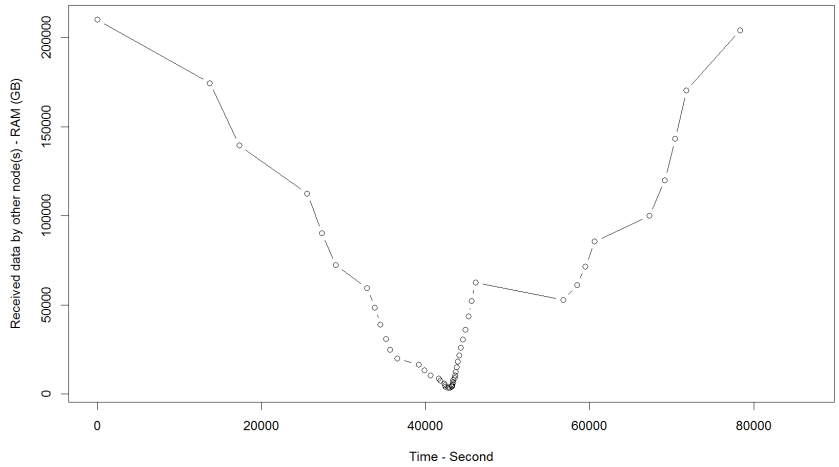

Figure I.25: How a cloud management system views RAM resources with a *relative* threshold value 20, the updates points are plotted a circles.

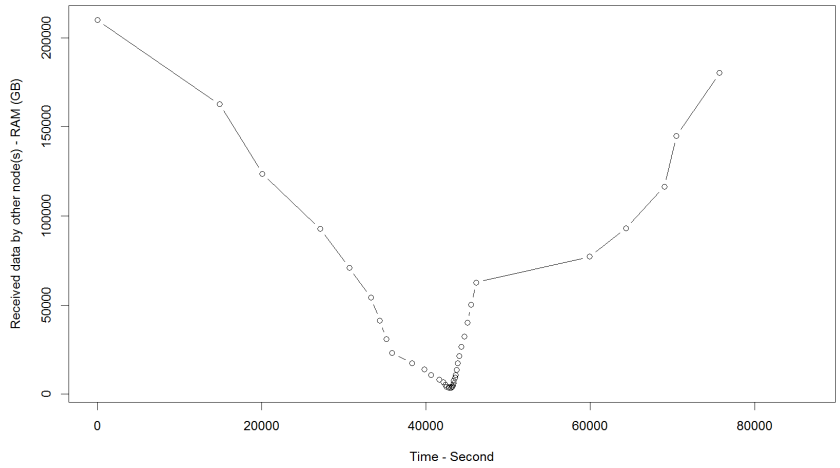

Figure I.26: How a cloud management system views RAM resources with a *relative* threshold value 25, the updates points are plotted a circles.

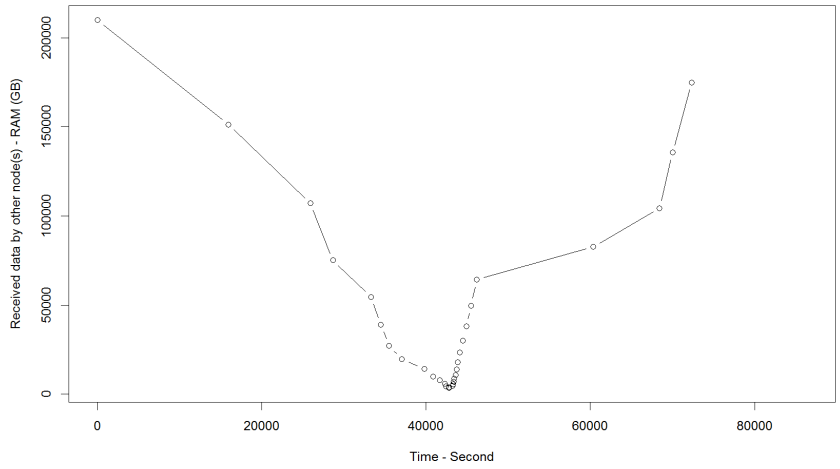

Figure I.27: How a cloud management system views RAM resources with a *relative* threshold value 30, the updates points are plotted a circles.

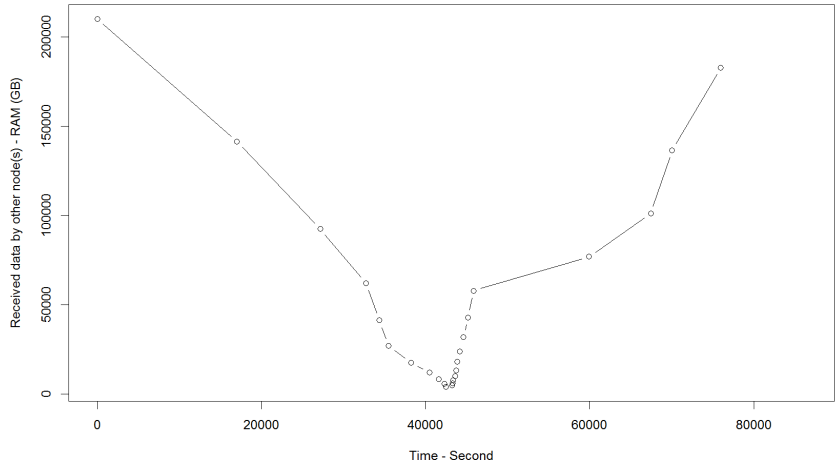

Figure I.28: How a cloud management system views RAM resources with a *relative* threshold value 35, the updates points are plotted a circles.

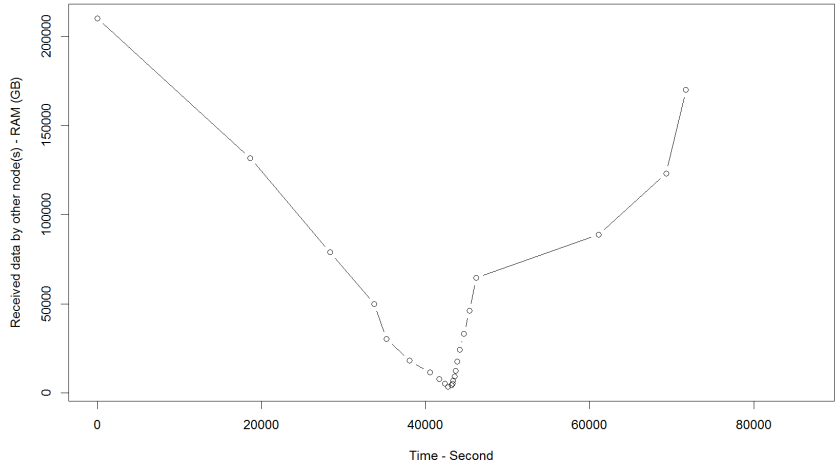

Figure I.29: How a cloud management system views RAM resources with a *relative* threshold value 40, the updates points are plotted a circles.

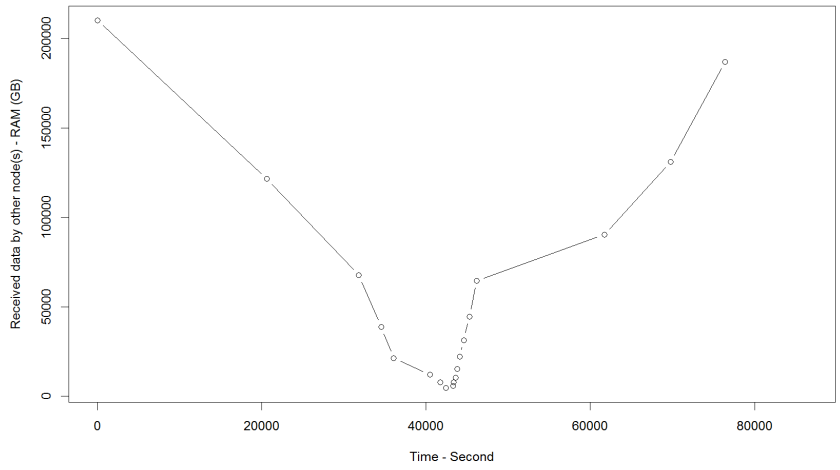

Figure I.30: How a cloud management system views RAM resources with a *relative* threshold value 45, the updates points are plotted a circles.

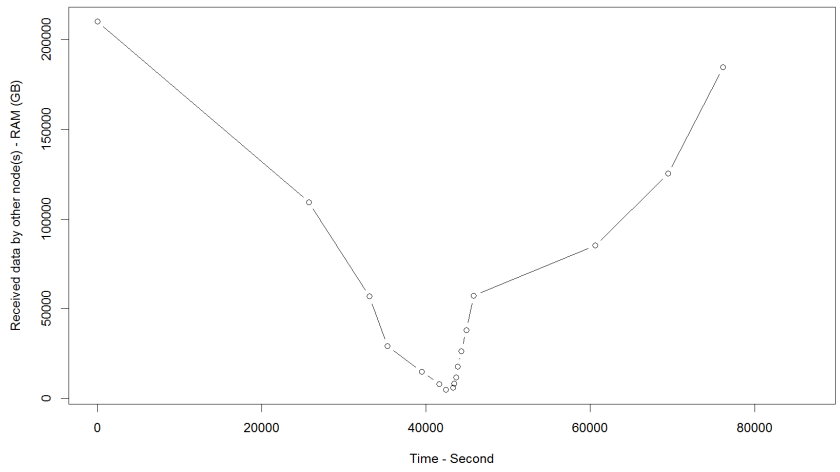

Figure I.31: How a cloud management system views RAM resources with a *relative* threshold value 50, the updates points are plotted a circles.

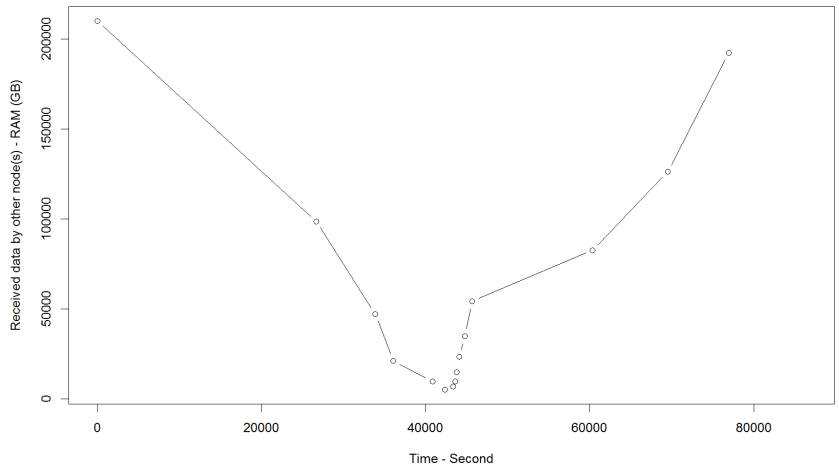

Figure I.32: How a cloud management system views RAM resources with a *relative* threshold value 55, the updates points are plotted a circles.

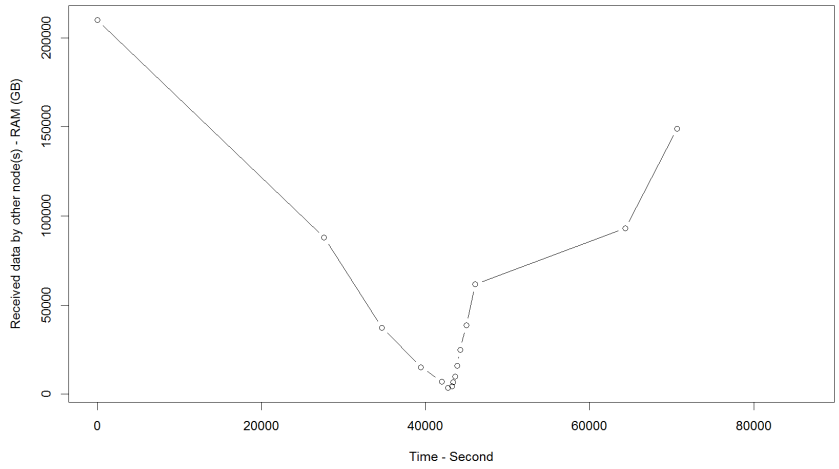

Figure I.33: How a cloud management system views RAM resources with a *relative* threshold value 60, the updates points are plotted a circles.

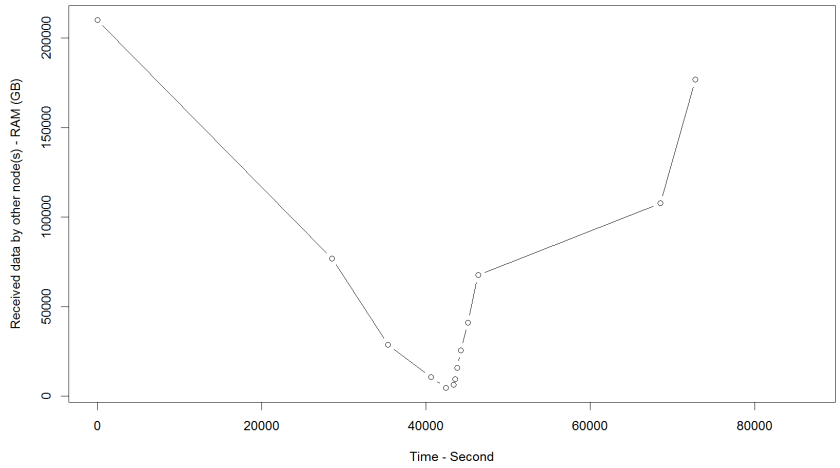

Figure I.34: How a cloud management system views RAM resources with a *relative* threshold value 65, the updates points are plotted a circles.

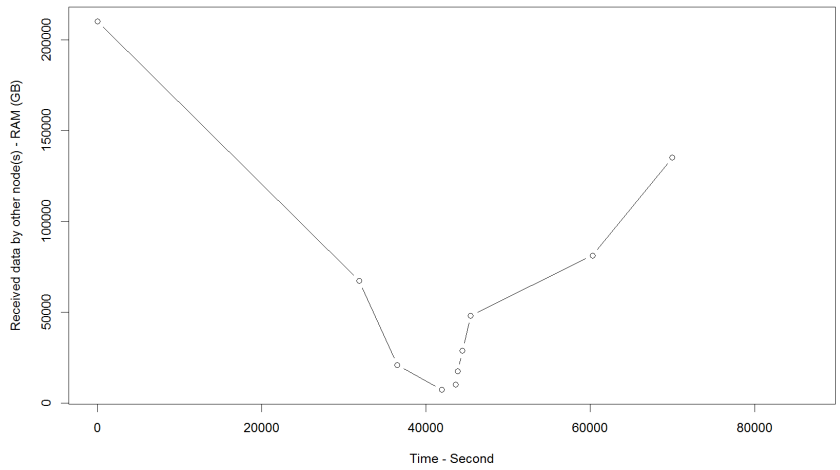

Figure I.35: How a cloud management system views RAM resources with a *relative* threshold value 70, the updates points are plotted a circles.

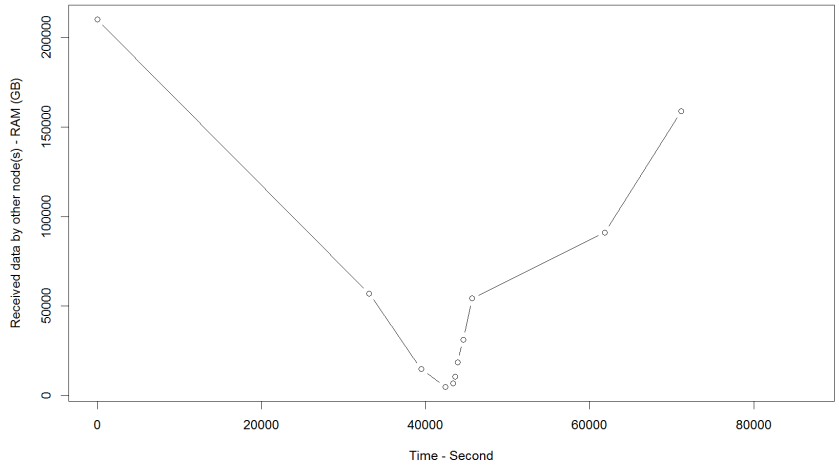

Figure I.36: How a cloud management system views RAM resources with a *relative* threshold value 75, the updates points are plotted a circles.

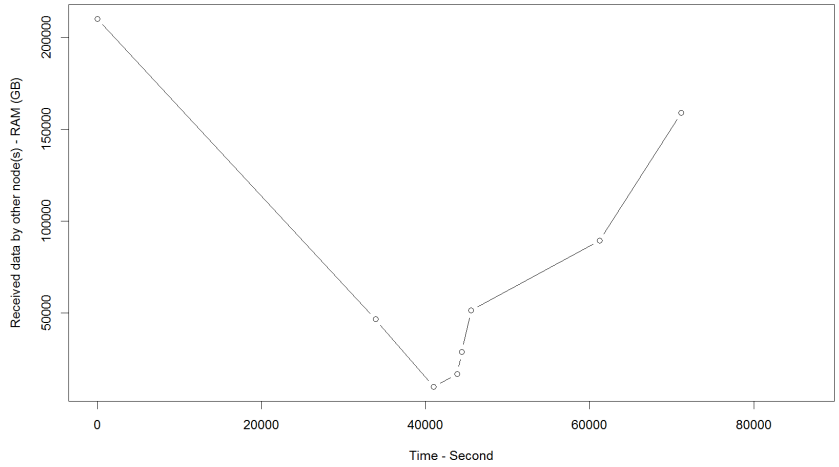

Figure I.37: How a cloud management system views RAM resources with a *relative* threshold value 80, the updates points are plotted a circles.

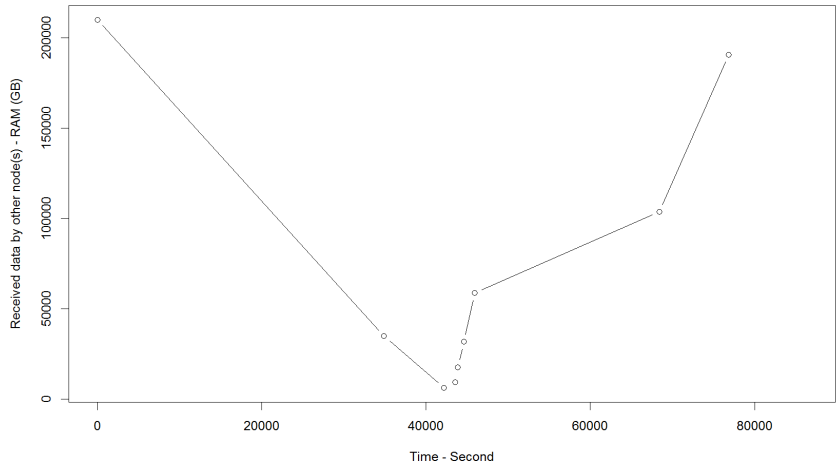

Figure I.38: How a cloud management system views RAM resources with a *relative* threshold value 85, the updates points are plotted a circles.

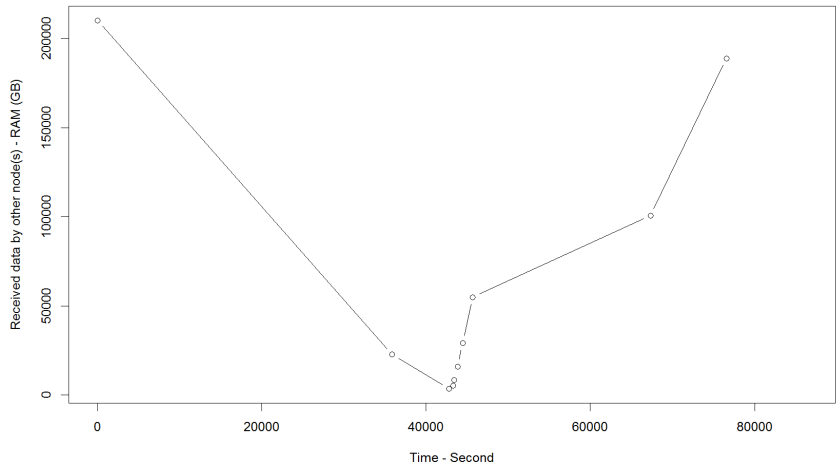

Figure I.39: How a cloud management system views RAM resources with a *relative* threshold value 90, the updates points are plotted a circles.

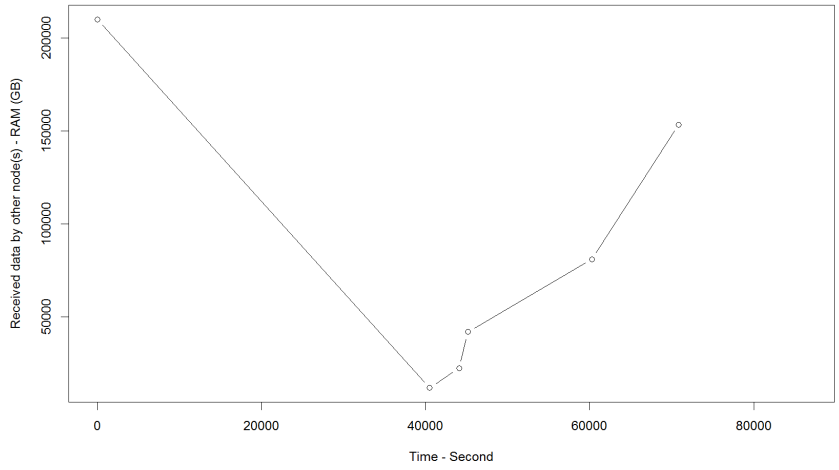

Figure I.40: How a cloud management system views RAM resources with a *relative* threshold value 95, the updates points are plotted a circles.

I.3 Data center's storage

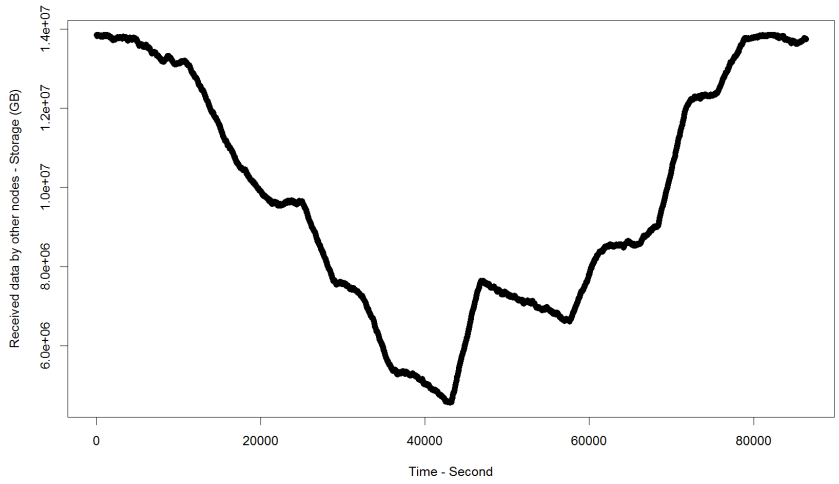

Figure I.41: Data center sample storage (GB) capacity per second

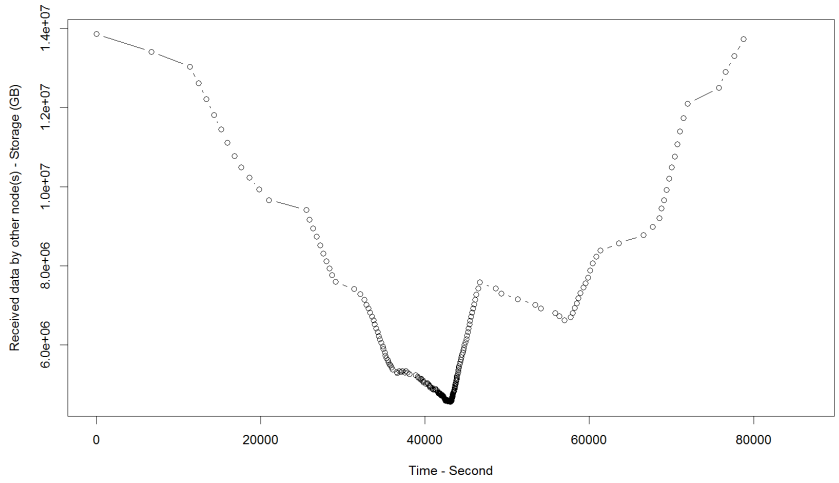

Figure I.42: How a cloud management system views storage resources with a *relative* threshold value 5, the updates points are plotted a circles.

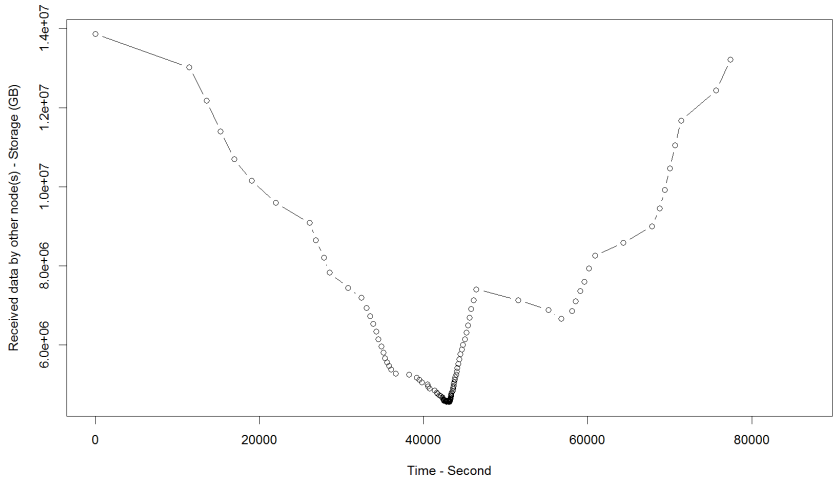

Figure I.43: How a cloud management system views storage resources with a *relative* threshold value 10, the updates points are plotted a circles.

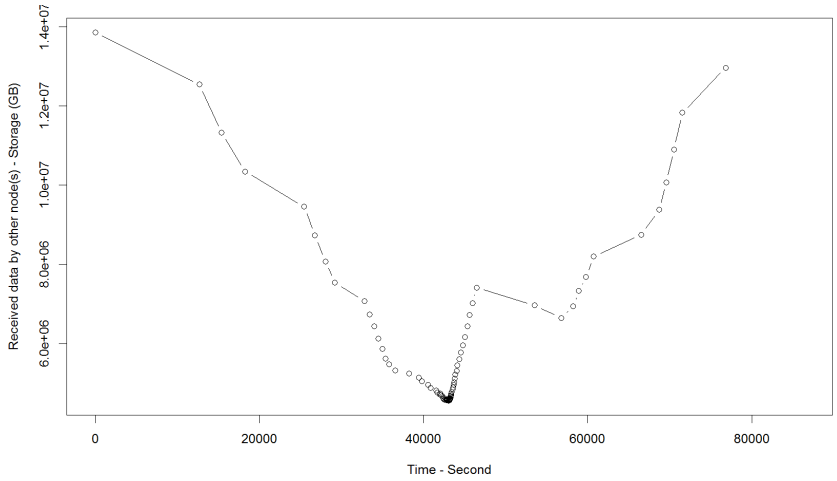

Figure I.44: How a cloud management system views storage resources with a *relative* threshold value 15, the updates points are plotted a circles.

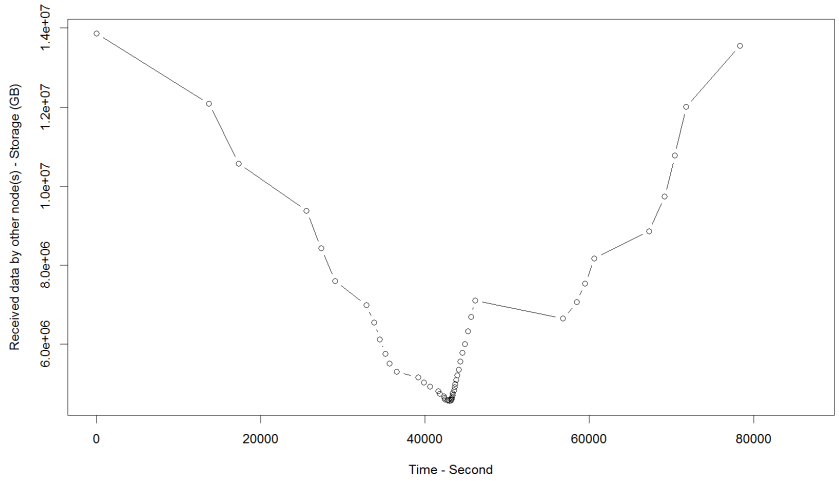

Figure I.45: How a cloud management system views storage resources with a *relative* threshold value 20, the updates points are plotted a circles.

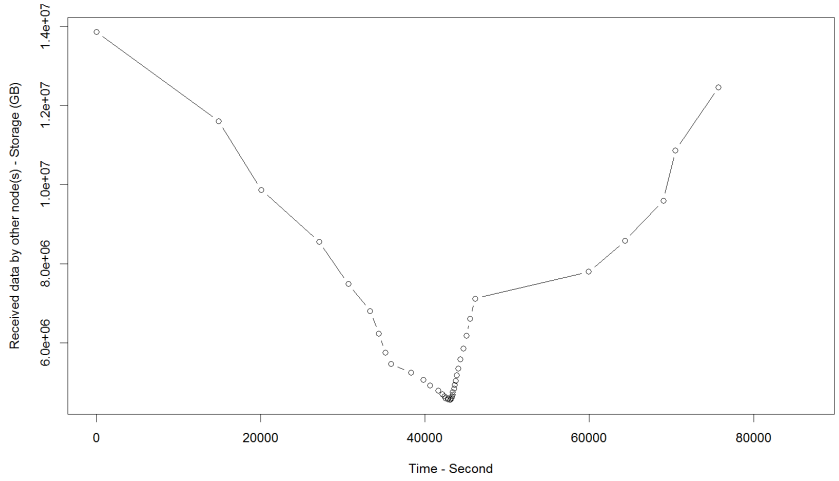

Figure I.46: How a cloud management system views storage resources with a *relative* threshold value 25, the updates points are plotted a circles.

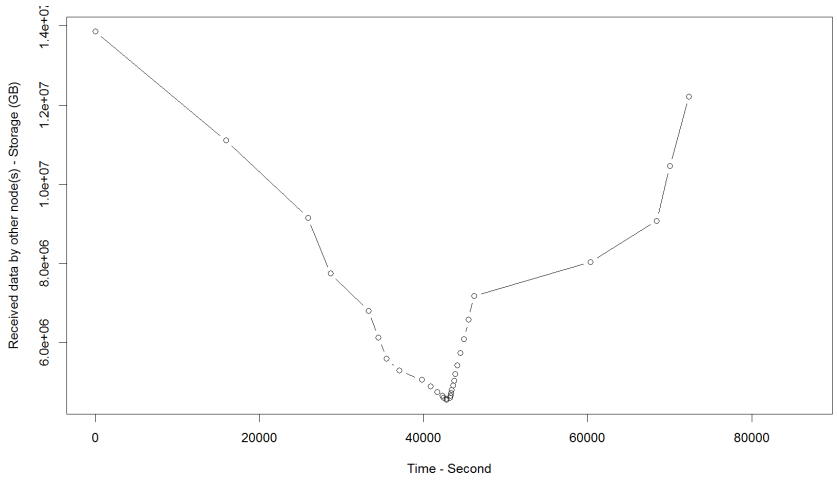

Figure I.47: How a cloud management system views storage resources with a *relative* threshold value 30, the updates points are plotted a circles.

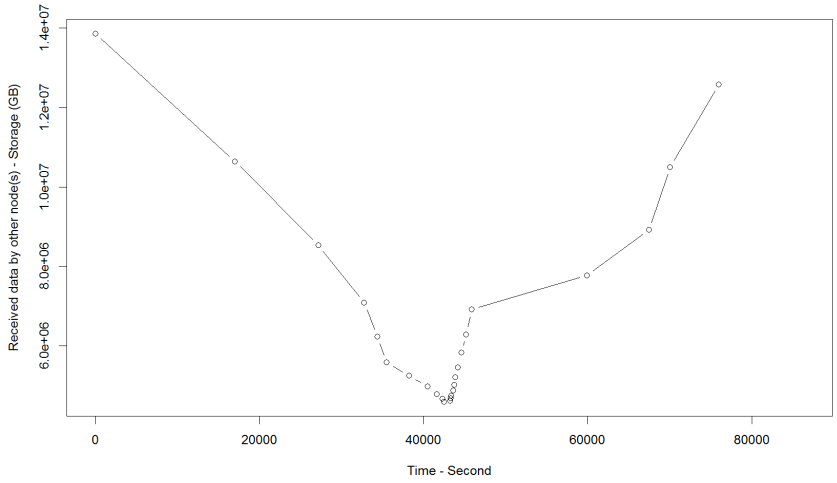

Figure I.48: How a cloud management system views storage resources with a *relative* threshold value 35, the updates points are plotted a circles.

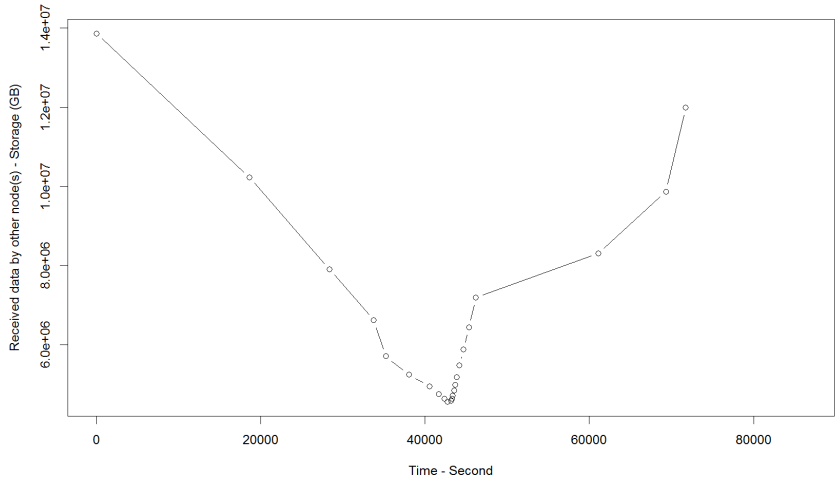

Figure I.49: How a cloud management system views storage resources with a *relative* threshold value 40, the updates points are plotted a circles.

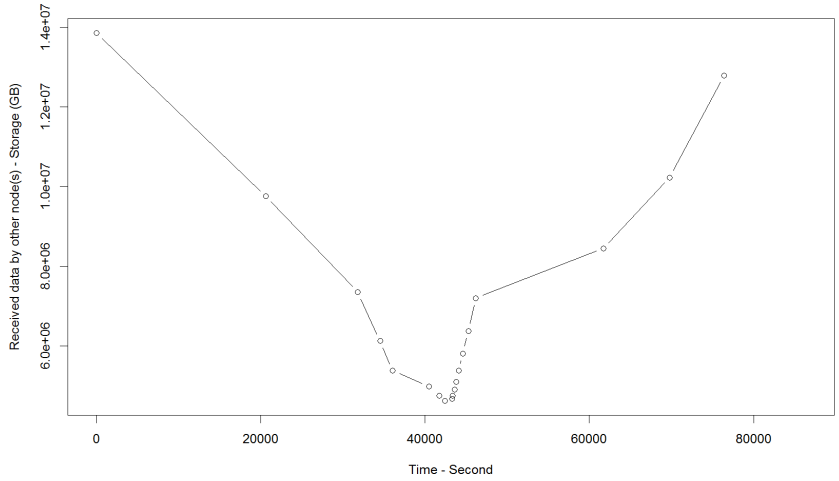

Figure I.50: How a cloud management system views storage resources with a *relative* threshold value 45, the updates points are plotted a circles.

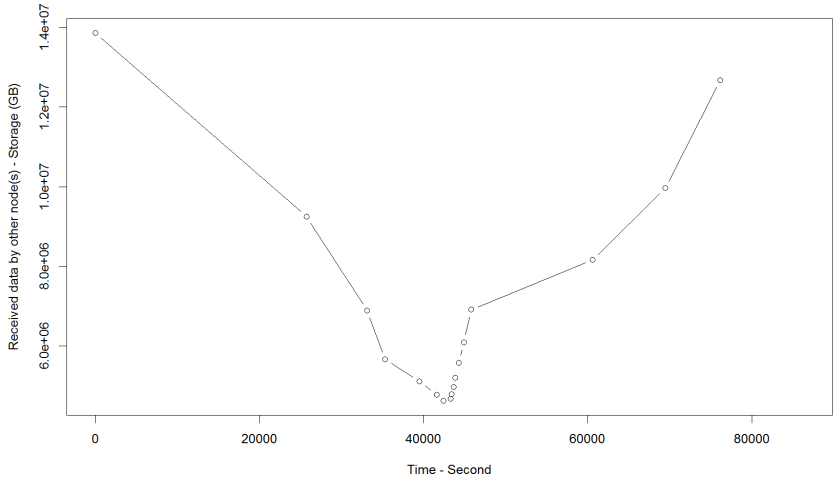

Figure I.51: How a cloud management system views storage resources with a *relative* threshold value 50, the updates points are plotted a circles.

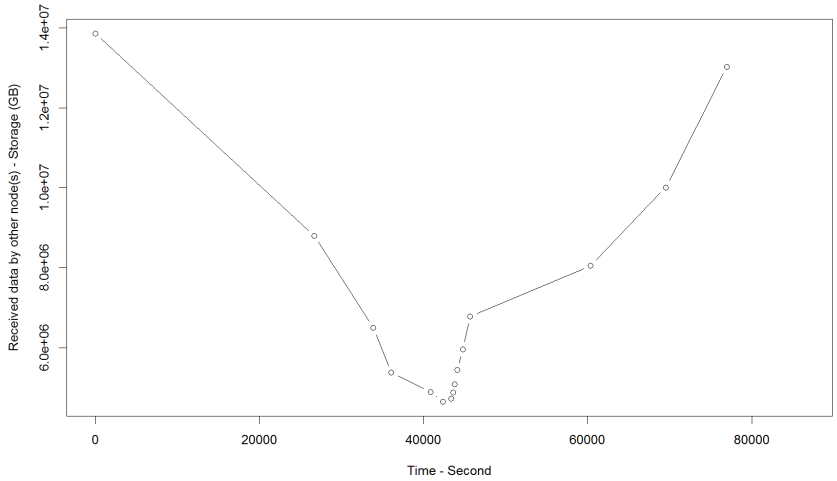

Figure I.52: How a cloud management system views storage resources with a *relative* threshold value 55, the updates points are plotted a circles.

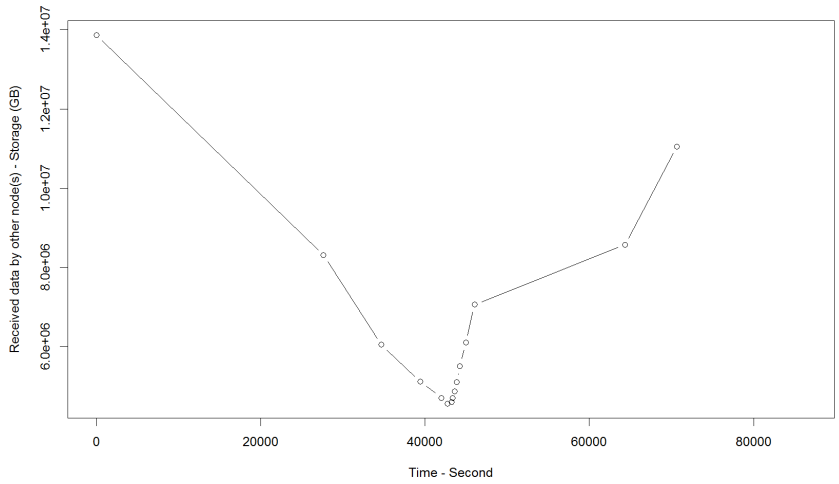

Figure I.53: How a cloud management system views storage resources with a *relative* threshold value 60, the updates points are plotted a circles.

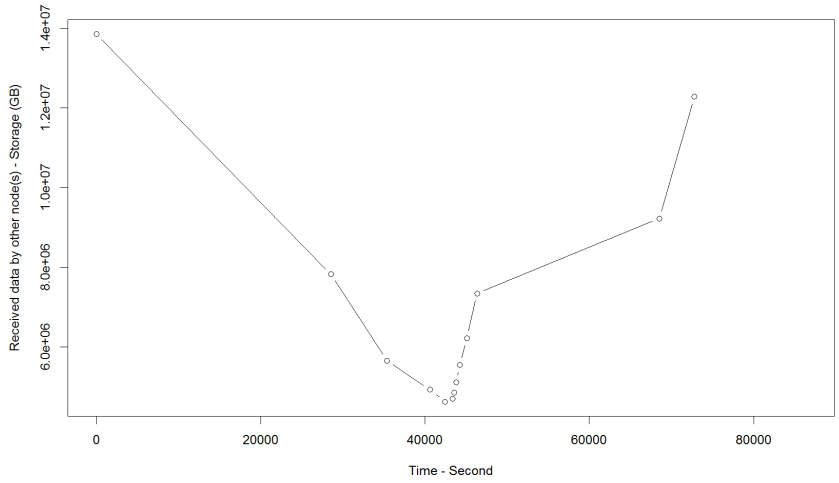

Figure I.54: How a cloud management system views storage resources with a *relative* threshold value 65, the updates points are plotted a circles.

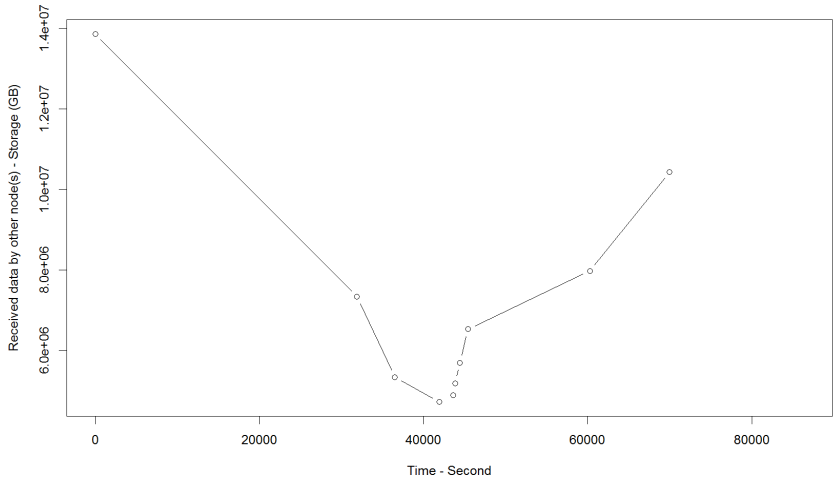

Figure I.55: How a cloud management system views storage resources with a *relative* threshold value 70, the updates points are plotted a circles.

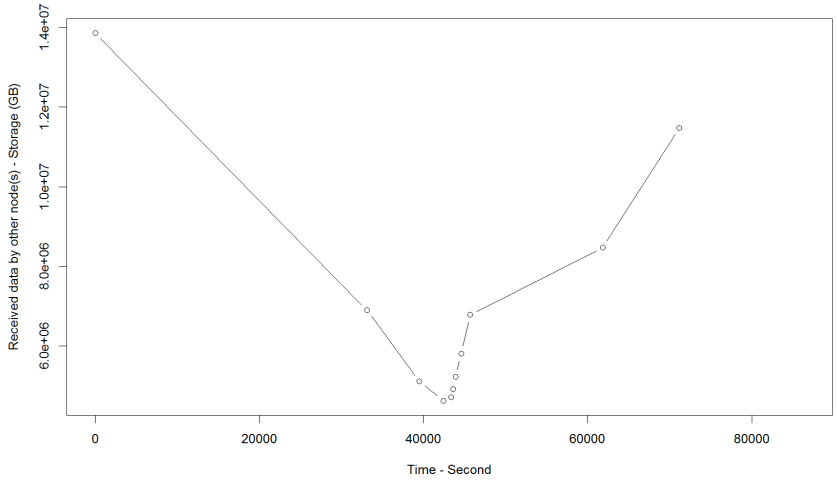

Figure I.56: How a cloud management system views storage resources with a *relative* threshold value 75, the updates points are plotted a circles.

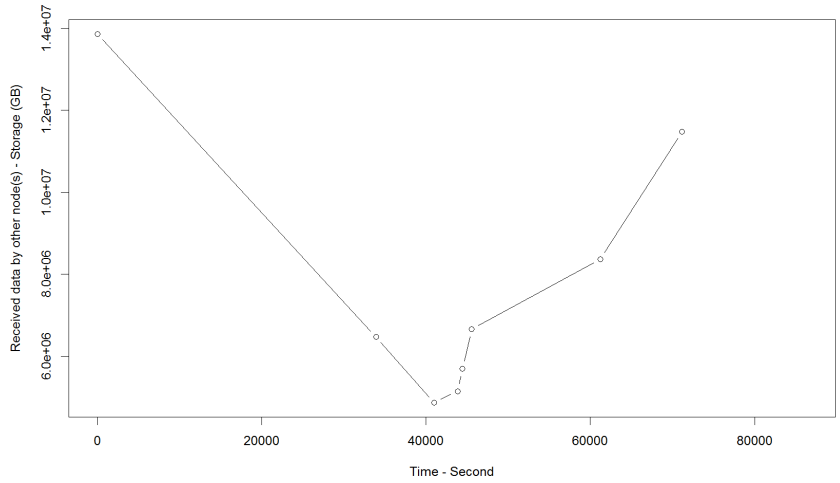

Figure I.57: How a cloud management system views storage resources with a *relative* threshold value 80, the updates points are plotted a circles.

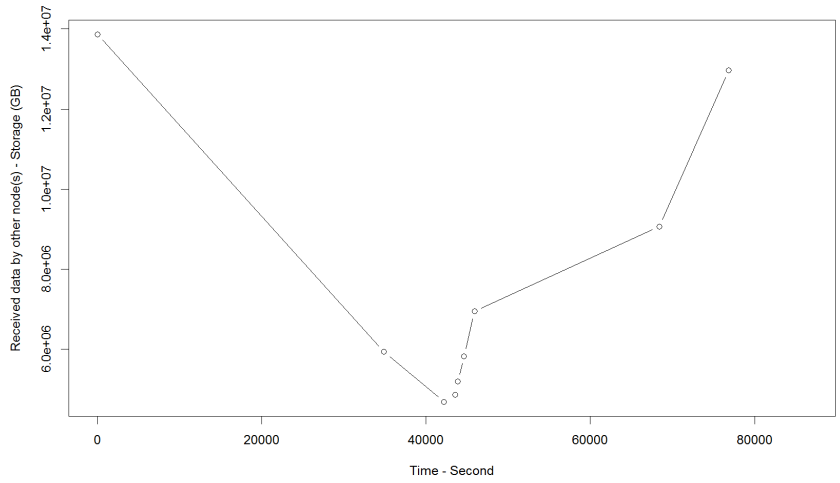

Figure I.58: How a cloud management system views storage resources with a *relative* threshold value 85, the updates points are plotted a circles.

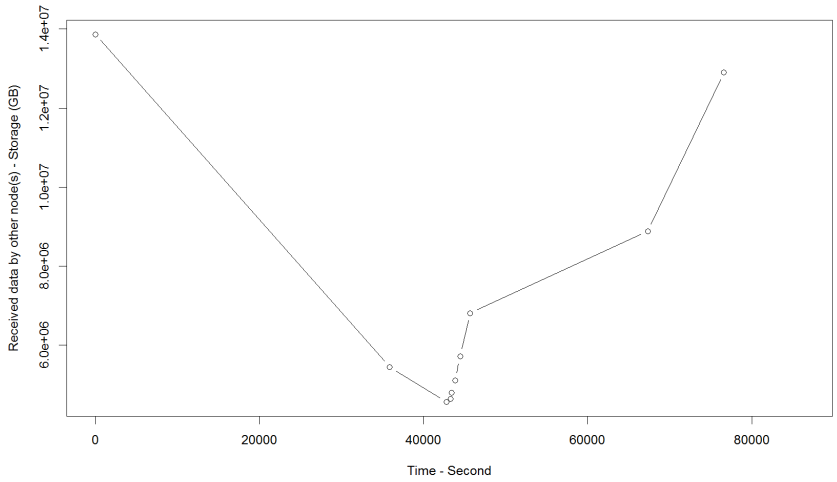

Figure I.59: How a cloud management system views storage resources with a *relative* threshold value 90, the updates points are plotted a circles.

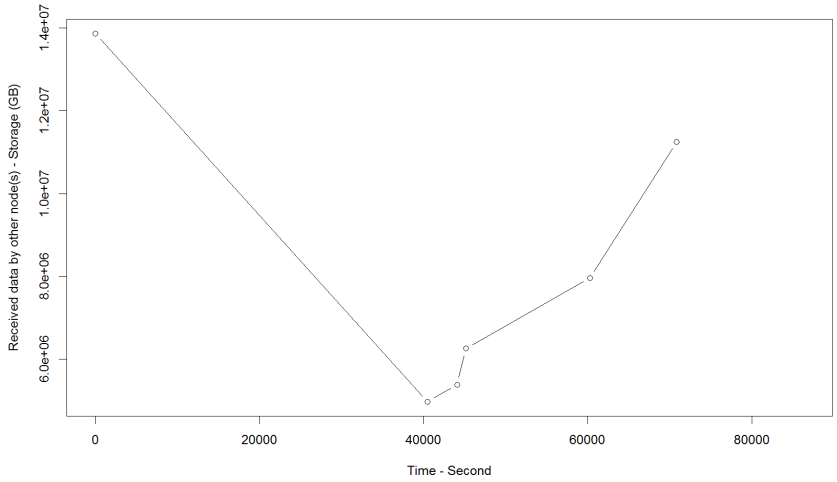

Figure I.60: How a cloud management system views storage resources with a *relative* threshold value 95, the updates points are plotted a circles.

Appendix J

Test results

In this appendix the results with a scale of **bytes per minute** for one round of test scenarios are given.

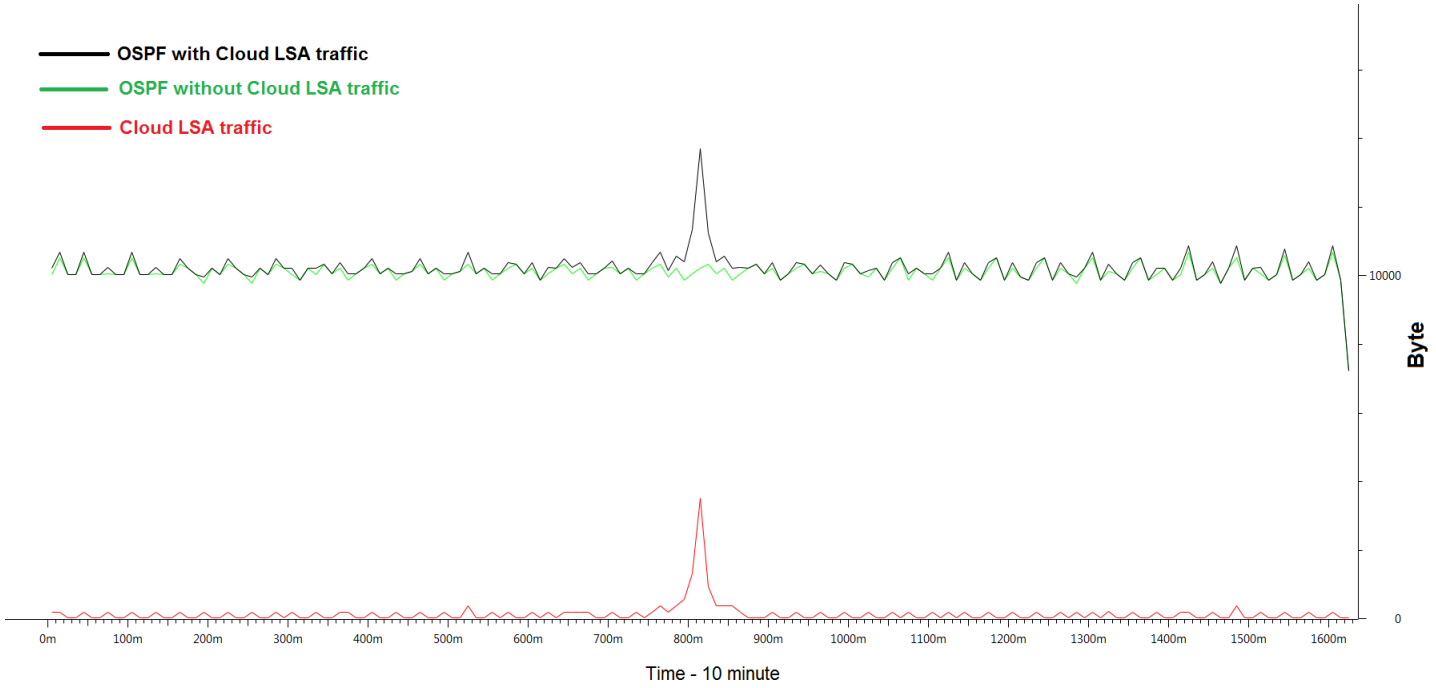

Figure J.1: The OSPF protocol traffic - bytes per 10 minute. First test scenario: One embedded data center in network. protocol traffic for OSPF without Cloud LSA, OSPF with Cloud LSA, and the Cloud LSA traffic

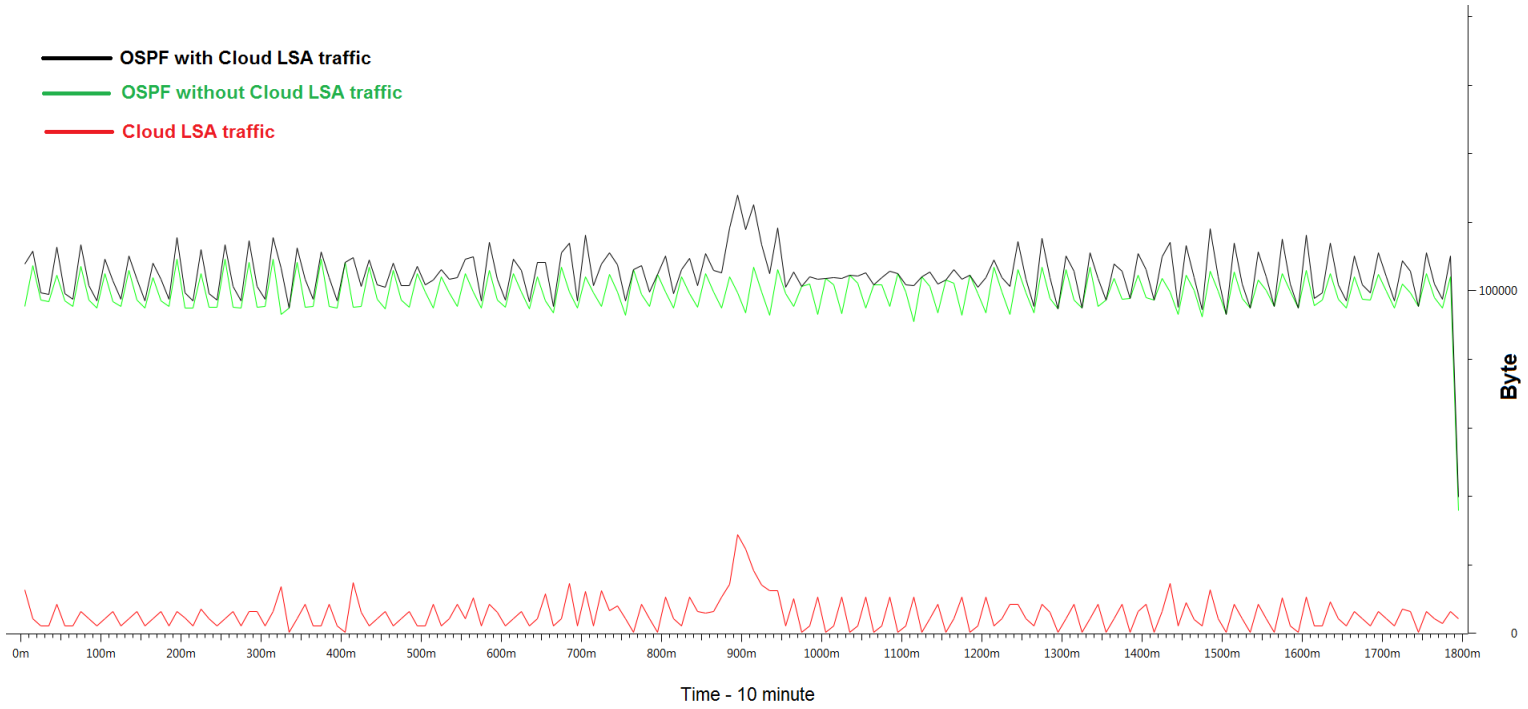

Figure J.2: The OSPF protocol traffic - bytes per 10 minute. Third test scenario: Six embedded data center in network. protocol traffic for OSPF without Cloud LSA, OSPF with Cloud LSA, and the Cloud LSA traffic.

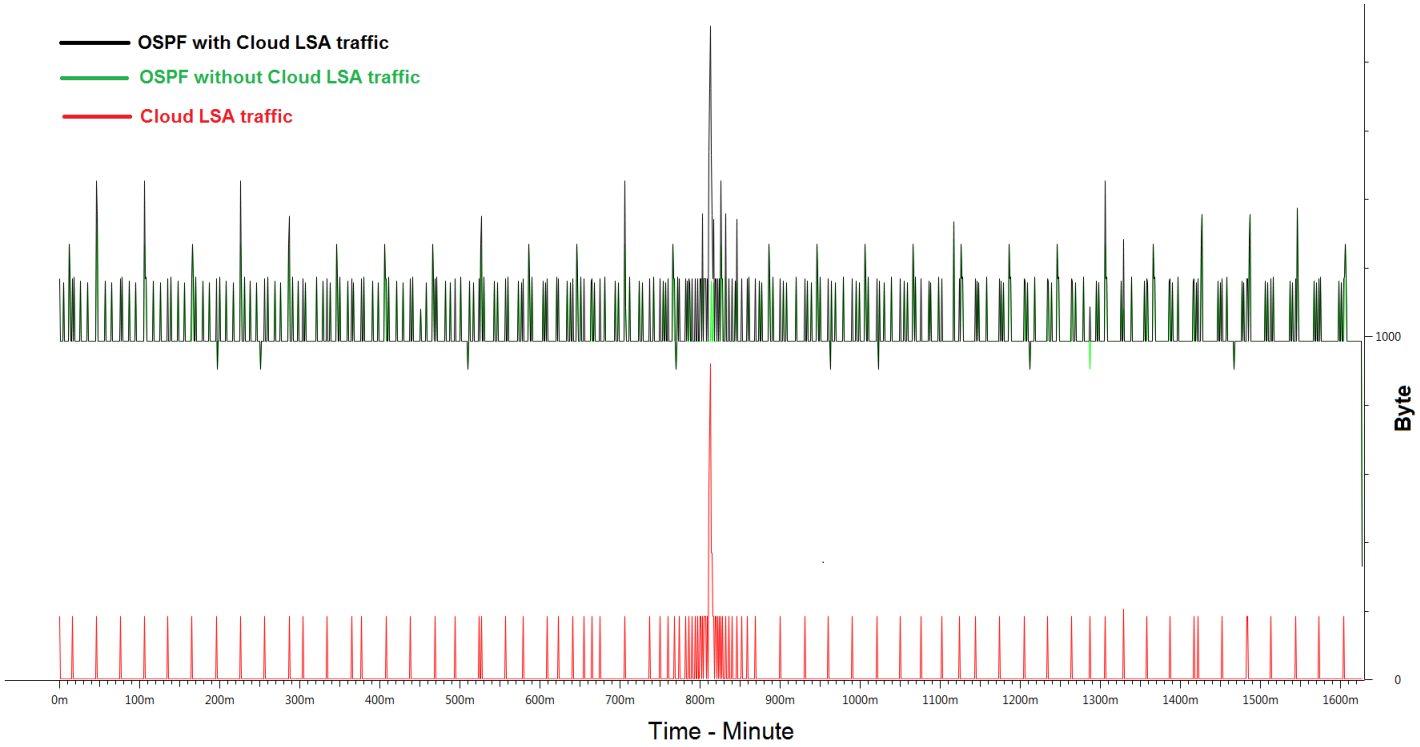

Figure J.3: The OSPF protocol traffic - Byte per minute . Test scenario one - one embedded data center in network. protocol traffic for OSPF without Cloud LSA, OSPF with Cloud LSA, and the Cloud LSA traffic

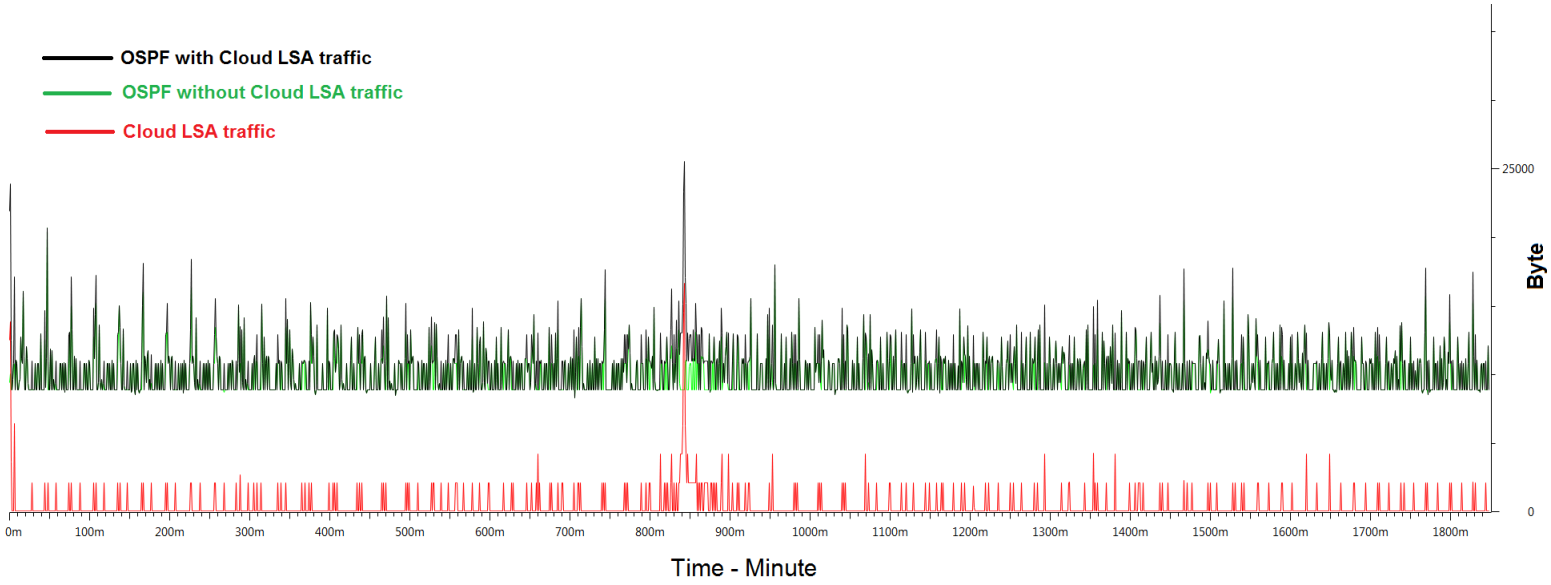

Figure J.4: The OSPF protocol traffic - Byte per minute . Test scenario two - tree embedded data center in network. protocol traffic for OSPF without Cloud LSA, OSPF with Cloud LSA, and the Cloud LSA traffic

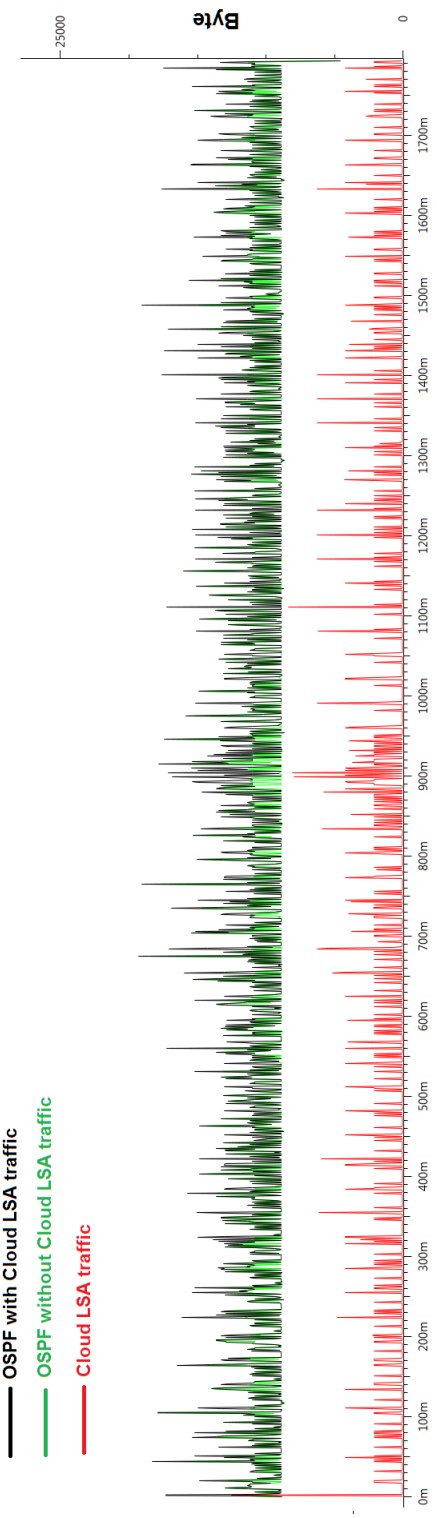

Figure J.5: The OSPF protocol traffic - Byte per minute . Test scenario tree - six embedded data center in network. protocol traffic for OSPF without Cloud LSA, OSPF with Cloud LSA, and the Cloud LSA traffic

TRITA-ICT-EX-2013:85

www.kth.se