NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF NATURAL SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS
DEPARTMENT OF ECONOMICS

POSTGRADUATE STUDIES PROGRAMME
MANAGEMENT AND ECONOMICS OF TELECOMMUNICATION NETWORKS

MASTER THESIS

The Use of Stock Options for Resource Allocation and Management on Content Delivery Networks

Matthias Filippos C. Doukas
Panagiotis S. Giannopoulos

Supervisor: **Stathes P. Hadjiefthymiades,** Professor

ATHENS

JULY 2019

MASTER THESIS

The Use of Stock Options for Resource Allocation and Management on Content Delivery Networks

Matthias Filippos C. Doukas

S.N.: ΜΟΠ 486

Panagiotis S. Giannopoulos

S.N.: ΜΟΠ 247

Supervisor: **Stathes P. Hadjiefthymiades,** Professor

JULY 2019

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία πραγματεύεται τη χρήση δικαιωμάτων προαίρεσης ως μία αποτελεσματική μέθοδο κατανομής και διαχείρισης πόρων, οι οποίοι διανέμονται από σύγχρονα δίκτυα διανομής περιεχομένων.

Επιχειρείται η αλληγορική συσχέτιση μεταξύ των επιστημών των Οικονομικών και της Πληροφορικής εξετάζοντας τη χρησιμότητα ενός καθαρά οικονομικού όρου - όπως είναι τα δικαιώματα προαίρεσης - για τη δημιουργία ενός προδραστικού μοντέλου διαχείρισης πόρων, το οποίο μπορεί να εφαρμοστεί άμεσα στο χώρο των δικτύων διανομής περιεχομένου, δηλαδή στην πλειοψηφία των υποδομών παρόχων περιεχομένου της καθημερινότητας της ψηφιακής εποχής.

Τα εισαγωγικά κεφάλαια παρουσιάζουν τον ορισμό των σύγχρονων δικτύων παροχής περιεχομένου και υπογραμμίζουν την ανάγκη για αποτελεσματική διαχείριση πόρων. Ταυτόχρονα, παρουσιάζεται η έννοια και η σημασία της παροχής ποιότητας υπηρεσιών στην παρούσα εποχή, μαζί με τις βασικές τεχνικές με τις οποίες αυτή επιτυγχάνεται στη σύγχρονη αγορά. Στη συνέχεια παρουσιάζεται το υπάρχον μοντέλο πρόβλεψης για την προσομοίωση της αγοράς, μαζί με τους μηχανισμούς και τα εμπλεκόμενα μέρη. Το μοντέλο αυτό χρησιμεύει ως θεμελιώδης ιδέα για την περαιτέρω έρευνα και το μοντέλο που προτείνεται στο πλαίσιο της εργασίας. Αναφέρονται επίσης οι βασικές έννοιες με τις οποίες ασχολείται η εργασία - δηλαδή οι ορισμοί της Δευτερεύουσας Αγοράς και των Δικαιωμάτων Προαίρεσης, οι οποίοι αναλύονται σε βάθος στα κεφάλαια 3 και 4 αντίστοιχα. Παρουσιάζεται η εξίσωση Black-Scholes για την αξιολόγηση των δικαιωμάτων προαίρεσης αγοράς μετοχών, στην οποία βασίζεται το προτεινόμενο μοντέλο, το οποίο με τη σειρά του παρουσιάζεται στα επόμενα κεφάλαια. Στο κεφάλαιο 5 αναλύονται τα σενάρια της αγοράς με βάση την ανάγκη χρήσης και εκμετάλλευσης των δικαιωμάτων προαίρεσης αγοράς μετοχών, υπογραμμίζοντας τη σημασία τους, ενώ στο κεφάλαιο 6 παρουσιάζονται οι βασικές μέθοδοι και πολιτικές κοστολόγησης και τιμολόγησης από την πλευρά των δικτύων διανομής περιεχομένου. Το κεφάλαιο 7 παρουσιάζει το προτεινόμενο μοντέλο έχοντας ως βάση όλες τις προαναφερθείσες έννοιες. Ο προτεινόμενος αλγόριθμος παρουσιάζεται βήμα προς βήμα και υπογραμμίζει τη διαφοροποίηση του μοντέλου σε σχέση με άλλα μοντέλα. Τα σενάρια προσομοίωσης του μοντέλου παρουσιάζονται στο κεφάλαιο 8, όπου καταγράφονται οι μετρήσεις και αναλύονται τα αποτελέσματα στο πλαίσιο της συμβολής του μοντέλου προς το στόχο της αποτελεσματικότερης διαχείρισης των πόρων στην αγορά των δικτύων διανομής περιεχομένου. Το κεφάλαιο 9 καταγράφει τα συμπεράσματα και παρουσιάζει ιδέες και ερωτήματα που μπορεί να αποτελέσουν αντικείμενο περαιτέρω μελέτης, ενώ στο τέλος της εργασίας αναφέρονται το βιβλιογραφικό έργο και άλλες πηγές, οι οποίες συνέβαλαν στην εκπόνηση της διπλωματικής εργασίας. Επίσης εμπεριέχεται ένα τριμερές υπόμνημα. Στο πρώτο μέρος παρουσιάζονται πολιτικές γνωστών παρόχων της εποχής για την κοστολόγηση πόρων. Το δεύτερο μέρος περιέχει τις βασικές συναρτήσεις του πηγαίου κώδικα Java του προγράμματος, το οποίο αναπτύχθηκε για τις προσομοιώσεις του προτεινόμενου μοντέλου ενώ το τρίτο μέρος περιέχει τον πίνακα τιμολόγησης του δικτύου διανομής περιεχομένου, όπως αυτός χρησιμοποιείται στις προσομοιώσεις.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Προδραστική Διαχείριση Πόρων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Δικαιώματα Προαίρεσης, Δευτερογενής Αγορά, Δίκτυα Διανομής Περιεχομένου, Δέσμευση Πόρων, Μοντέλο Black-Scholes

ABSTRACT

This thesis presents and examines the use of Stock Options as an efficient allocation and management method of resources, which are distributed from modern infrastructure Content Delivery Networks.

The allegorical correlation between the scientific fields of Economics and Computer Science is attempted by examining the usefulness of a purely financial term - such as Stock Options - for the creation of a proactive resource management model, which can be directly applied to content distribution networks, namely the majority of content providers' infrastructures in the everyday life of the digital age.

The introductory chapters present the definition of modern Content Delivery Networks and highlight the need for efficient resource management. The concept and the importance of providing service quality in the present era are also presented, along with the basic techniques by which it is achieved in the modern CDN ecosystem. Next, an existing resource prediction model of the market's simulation is presented, along with its mechanisms and the parties involved. This model serves as the fundamental idea on our further research and our own proposed model. The basic concepts that the thesis is deeply concerned with - namely the definitions of the Secondary Market and the Stock Options - are also introduced.

The core concepts, namely the Secondary Market and the Stock Options are analyzed in detail in chapters 3 and 4 respectively. We present the Black-Scholes options pricing formula for the evaluation of stock options, based on which the proposed model of work is presented, which in turn is presented in the following chapters. Chapter 5 analyzes market scenarios based on the need to use and exploit stock options, highlighting their importance, while in chapter 6 fundamental CDN cost planning pricing methods and policies are presented.

Chapter 7 presents the proposed model on the basis of all the previously mentioned concepts. The proposed algorithm is laid out step by step and highlights the model's differentiation compared to other models. The conversion of the algorithm into actual, scenarios of our proposed framework is presented in chapter 8, where the metrics are presented and the results of the simulated scenarios are analyzed in the context of the model's contribution towards the goal of more efficient resource management in the area of Content Delivery Networks. Chapter 9 serves as the conclusion of the thesis and presents ideas and questions that may be the subject of further study, while at the end of the thesis references of the literature work and of all other sources that contributed to the creation of the thesis are recorded.

There is also a three-part annex. In the first part, examples of cost pricing policies of real-life CDN's are presented. The second part contains the fundamental functions of the Java source code that was developed for the simulation of the proposed model. Finally, the third part contains the CDN pricing table used in the simulation.

SUBJECT AREA: Predictive Resource Allocation

KEYWORDS: Stock Options, Secondary Market, Content Delivery Networks, Resource Allocation, Black-Scholes Model

This thesis is dedicated to our families and friends.

ACKNOWLEDGEMENTS

We would like to thank Professor Stathes Hadjiefthymiades and PhD candidate Mr. Elias Vathias for their guidance and assistance throughout the lengthy period of working on this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This thesis was developed as the culmination of our postgraduate studies in the Interdepartmental Postgraduate Programme of Management and Economics of Telecommunication Networks, which is co-organized by the Department of Informatics and Telecommunications and the Department of Economics of the National and Kapodistrian University of Athens. The authorship of the thesis required study on both economical and computer science field of work, especially in developing the proposed model framework algorithm and its subsequent conversion into the software application which serves as the simulation environment of the proposed model. In its entirety, the thesis required months of work and was completed in July 2019 in Athens, Greece.

1. INTRODUCTION

1.1 The need for Efficient Resource Management in the area of Content Delivery Networks

Content delivery networks (CDN) are the transparent backbone of the Internet's digital world, being in charge of content delivery across multiple geographical areas. The transparency is highlighted because end users interact with CDNs on a daily basis; when reading articles on news sites, shopping online, watching YouTube videos or perusing social media feeds. No matter the specific type of content, chances are that CDNs are behind every character of text, every image pixel and every movie frame that gets delivered to your PC and mobile browser.

Their importance of CDNs is recognized when regarding the issue they are designed to solve: latency. Latency is described as the delay of the content's transmission from its source to the destination client that requested to access it. That delay interval is affected by a number of factors, many being specific to a given web page. In all cases however, the delay duration is impacted by the physical distance between the source and the destination. A CDN's mission is to virtually shorten that physical distance, the goal being to improve both speed and performance while at the same time maintaining the content's integrity.

A CDN serves its content to multiple Origin Servers (OS). Origin Servers are scattered across different geographical locations and aim to serve the content to the end users that are a short distance away and have made a request to the CDN. Therefore, OS operate both as first-level, internal clients of the CDN and as transparent servers of content to the end users that interact with the CDN.

The architecture of this ecosystem establishes the need for efficient cooperation between CDNs and Origin Servers. Since by definition resources are finite and scarce, the creation of an optimal framework that promotes the notion of efficient "Resource Allocation and Management" in the area of Content Delivery Networks is of highest importance.

1.2 Quality of Service in Content Delivery Networks

Studies have shown that even 1 second of network delay causes 11% decline in websites traffic and 16% decline in customer satisfaction indexes [3]. It is therefore important to outline certain aspects about the Quality of Service (QoS) regarding resources utilization in the area of Content Delivery Networks.

CDNs have been designed to be efficient and effective during periods of high traffic or even during sudden rise of demand in an otherwise period of stable traffic load. Most major commercial CDNs consist of a complex of thousand of web servers so that millions of end users are able to access the same content without performance degradation.

The goal of CDNs is to serve their content to the end users in a way where high performance, availability and efficiency is achieved. Due to the fact that the content is served through a farm of servers the traffic does not flow via a single server but it is distributed and therefore 'offloading' and resource –and cost- reservation is also achieved.

Furthermore it should be noted that due to the decentralized architecture adopted in Content Delivery Network topologies, the network has secured –besides the fast access

to the content data provided to the end users- its own integrity against network attacks such as Distributed Denial of Service (DDoS). Such kind of attacks aim to take out a main server by flooding it with a high number of content requests. Using a farm of multiple web servers in a CDN prevents a single point of failure in the network because the traffic is handled by all of them -using load balancing techniques- and subsequently every new request for content can be handled from any member server of the CDN depending on which member is the most unused at that particular moment.

In order for the Content Delivery Networks to ensure Quality of their Services they use foundational elements and techniques. Quality is achieved once the following goals are met:

- Speed Improvement
- Fault Tolerance
- Scalability
- Server Load Reduction
- High Availability

The following list presents specific methods and techniques are used by CDNs in order to achieve and improve Quality of their services:

- Caching Servers
- Reverse Proxy Server
- Content Replication
- Anycasting

Each one of the above is described in the following subsections of this chapter.

1.2.1 Caching Servers

Caching Servers are responsible for saving and fetching temporarily saved files. Their main goal is to improve the loading time of a resource (i.e. of a webpage) and reduce the bandwidth consumption. Each caching server is consisted of multiple storage units and large volume of RAM resources [3].

1.2.2 Reverse Proxy Server

When a CDN needs to provide a large volume of data to a large number of clients, it can reduce the load to the OS (Origin Servers) by saving the data to Caching Serves or other OSs. A server with such a role in this kind of topology is called Reverse Proxy Server. Contrary to Caching Servers, which are placed close to the clients in order to reduce both the required bandwidth and the response time, a Reverse Proxy Server is placed close to the Origin Servers which are on the edge of the network (also called Edge Servers). The topology of using a Reverse Proxy Server in a CDN is presented in the following image [5].

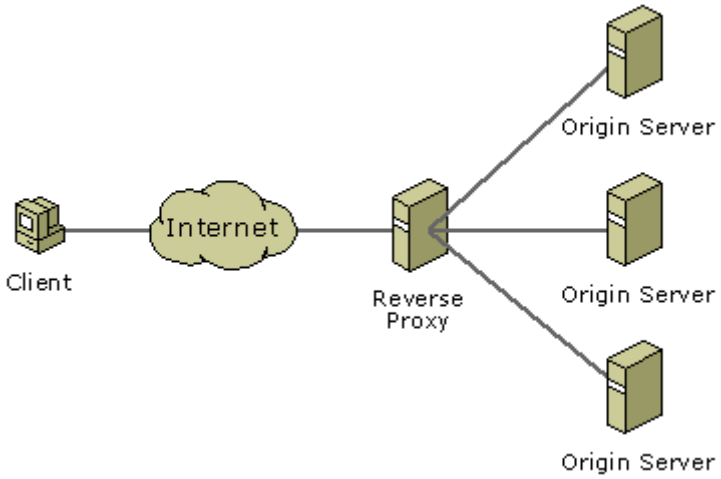

Figure 1: Reverse Proxy Server

The use of a Reverse Proxy Server helps the CDN in both the network's security and the traffic load balancing between its servers. It is an optimal solution since the Reverse Proxy becomes a predefined point from which all incoming traffic volume passes, therefore providing the CDN with a clear view of the traffic in the network and subsequently with the opportunity of applying load balancing by distributing the incoming clients' requests to the appropriate Origin Server, depending on each OS's workload at that moment.

1.2.3 Content Replication

Content Replication is a technique which aims to improve the access time to the data provided by the CDN's Origin Servers. Reducing delays will also improve the overall availability of the network's content. For example, instead of keeping a particular webpage on a single server, most CDNs are consisted of a large number of edge servers scattered in different locations all over the world, with the content of this webpage replicated and kept in each and every one of them, ready to be served to a client at any moment. So if a central server changes the content of the webpage, then by using replication methods, the content of the edge servers will also be updated. This technique improves the network's overall efficiency by serving client requests via multiple edge servers and not a single, centrally placed server.

1.2.4 Anycasting

Anycasting can be described as a network addressing and routing method where the sender's requests are routed to any available destination node, which are selected based on certain criteria. Selective routing in a network using Anycasting provides tolerance of higher workloads and protection against DDoS attacks and other security threats [6].

In a CDN, Anycasting usually routes incoming traffic to the nearest server with the ability to handle the requests rapidly and effectively.

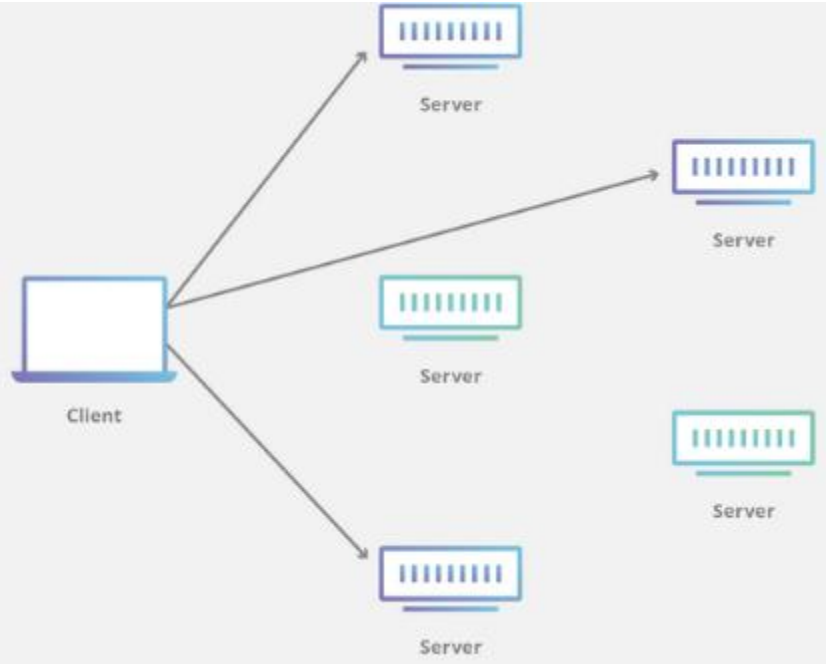

Figure 2: Anycasting

2. EXISTING RESOURCE PREDICTION FRAMEWORK

The topic of effective resource prediction and allocation has been the subject of several papers. Research of E. Vathias, D. Nikolopoulos, S. Hadjiefthymiades proposes a complete framework with a predictive reservation mechanism and also establishes the notion that the modern capital market can be used as metaphor in the ecosystem of Content Delivery Networks [1] [7].

2.1 Predictive Reservation Scheme

The architecture of the resource allocation prediction model of a CDN is presented in the following figure [2]. The adopted PRS (Predictive Reservation Scheme) model (Fig. 3) aims to calculate the traffic of the requested resources (i.e. Bandwidth or Disk Space) from CDN's clients as accurately as possible.

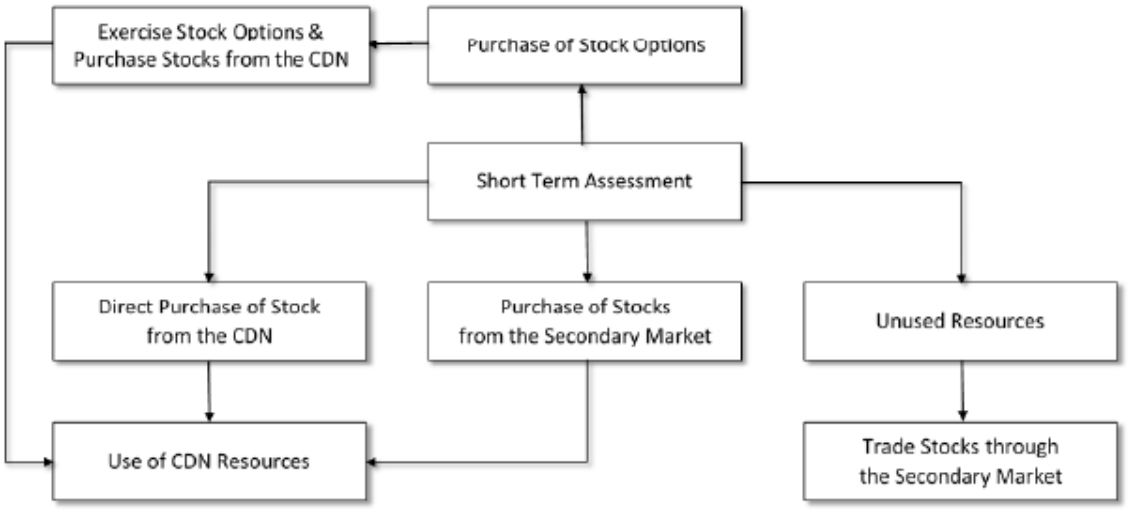

Figure 3: **Predictive Reservation Scheme (PRS)**

An accurate prediction of requested resources allows the CDN to allocate resources for an Origin Server (OS) in advance and thus, respond to the constantly fluctuating requested total workload more effectively. Resources can be requested but they can also be returned to the CDN in order to be used from other clients. This requires a tool that can be used for the exchange of resources and improve the durability of the prediction mechanism, in cases of misses in the prediction process. Such kind of tool is the Secondary Market (SM).

There is always the possibility that an Origin Server has already bought a certain amount of resources from the CDN based on its predicted needs. However, a sudden decline of its requested traffic load leaves this OS with a surplus of resources. The Secondary Market is the place where clients can sell their surplus of resources. These unused resources, once made available in the SM, can be bought by other clients who are in need of resources because of a miss in the prediction of their respective needs.

These clients can enter the Secondary Market and buy the resources directly from the sellers without the interference of the CDN.

The amount of exchanged resources is finite and relatively limited, therefore the CDN needs to adjust and manage the allocation process carefully and as much accurately to its clients' needs as possible. The existing model is based on two mechanisms for the maximization of resources' usage. The first and most important mechanism is the Secondary Market (SM, Fig. 4) where the exchange of resources between Origin Servers occurs. It is an alternative way for the OS to gain the needed amount of resources in case buying directly from the CDN is not feasible. In addition, there is a second entity, called feedback mechanism, which allows the OS to use its personal, unused resources in a later time, whenever it considers that necessary.

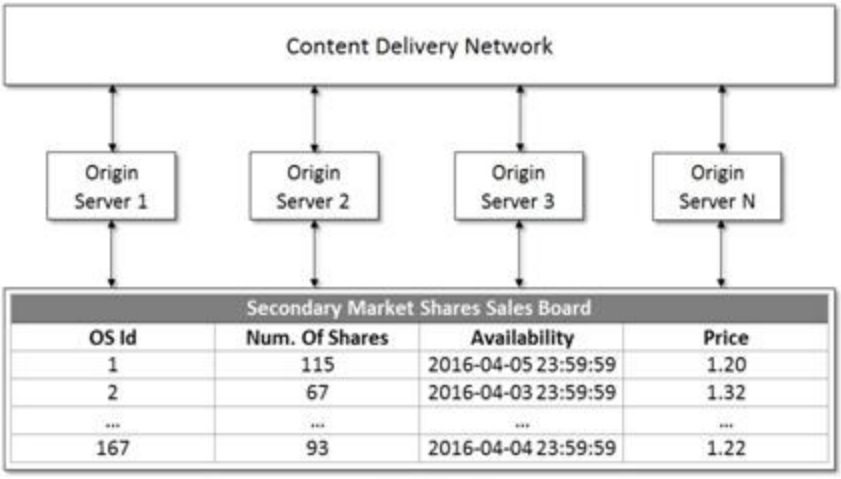

Figure 4: **Secondary Market**

2.2 Prediction Mechanism

The prediction mechanism is the first stage of the the PRS and is based on Kernel Regression Estimators (KREs) and other complementary techniques, which are described in the following paragraphs.

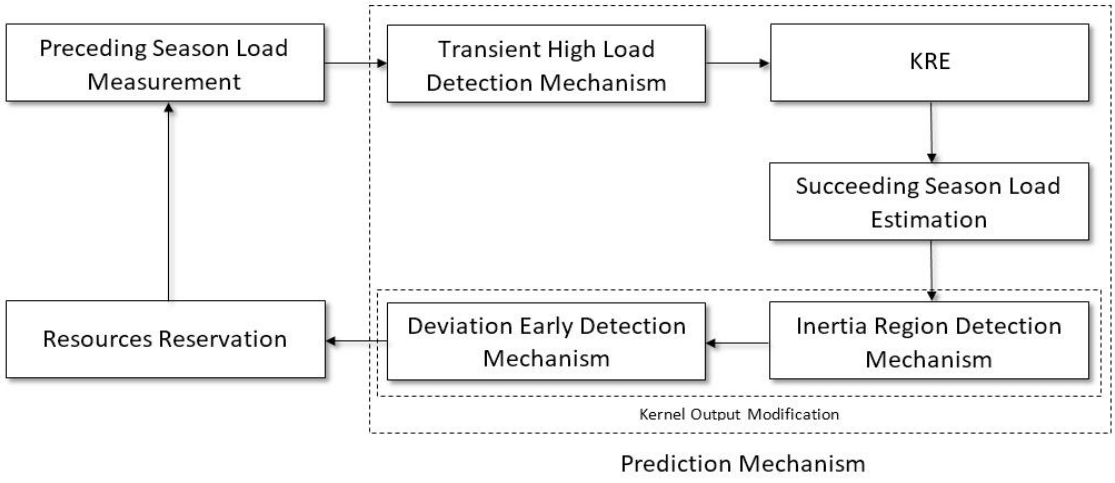

Figure 5: **Prediction Mechanism**

2.2.1 Transient High Load Detection Mechanism (THLDM)

This mechanism detects the high load that can be found in a network from phenomena such as Flash Crowds or Distributed Denial of Service (DDoS) attacks. Consequently both normal and malicious phenomena can be detected. When something like that is observed, the respective loads are exempt from the next prediction for the allocation of resources.

2.2.2 Kernel Regression Estimators

Kernel Regression Estimators aim to find a non-linear correlation between two random variables x and y. They are techniques which allow the modeling of the load with high level of precision in a specific timeframe usually expressed in hours. In the context of modeling the CDN-OS market the Gaussian Kernel Regression can be used, namely a linear algorithm with Gaussian Kernel [1], expressed as follows:

$$y^* = \frac{\sum_{i=1}^{N} K(x^*, x_i) y_i}{\sum_{i=1}^{N} K(x^*, x_i)}$$

In the above equation **Xi** is defined as the i-th data input and **Yi** as the i-th data output. Furthermore, **K** symbolizes the Kernel equation, **x*** is a query point and **y*** is the subsequent outcome. The Gaussian Kernel is defined as follows:

$$K(x^*, x_i) = exp\left(-\frac{(x^* - x_i)^2}{2b^2}\right)$$

The combination of both aforementioned equations offers the way of smoothing the data by providing a formula which can determine the load curve with greater success.

The result of using this Kernel Regression Estimator is a better filtered model of the expected load in time because it takes into account the produced and constantly revised function of the load in time.

2.2.3 Inertia Region Detection Mechanism (IRDM)

The Inertia Region Detection Mechanism modifies the outcome of the Kernel Regression Estimator by including the negative points, in which the load overcomes its maximum possible value [1].

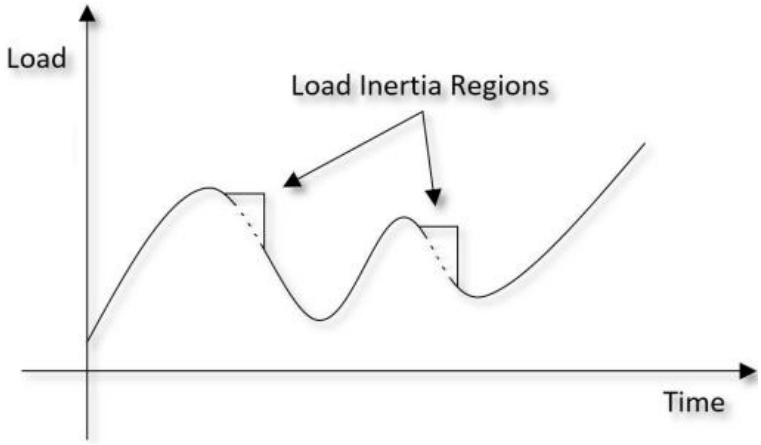

Figure 6: **Curve with Load Inertia Regions**

2.2.4 Deviation Early Detection Mechanism (DEDM)

The Deviation Early Detection Mechanism aims to track in time possible deviations in the prediction mechanism. More specifically, it observes in sequential periods of time whether the current load is similar to the load predicted by the prediction mechanism [1].

Based on this comparison, the DEDM modifies accordingly the amount of allocated resources of the next period of time by modifying the output of the KRE of that time.

$$CF = 1 + \frac{A_i - R_i}{R_i}$$

As **CF** the Calibration Factor is symbolized, namely the analogy of actually (**A**) needed resources compared to the reserved (**R**) resources of the **i**-th time frame.

As a result, the following equation can be observed:

$$\widehat{R}_{i+1} = CF * R_{i+1}$$

where the reserved resources for the i+1-th time frame are actually modified as the product of the Calibration Factor and the –until then- amount of reserved resources, as they were determined by the KRE.

2.2.5 Initial Resource Reservation Monitoring Mechanism (IRRMM)

The Initial Resource Reservation Monitoring Mechanism can be used only during the initial period of use of the prediction mechanism when there are no previous data available [1]. In that case, the IRRMM allocates resources only for the first day of use instead of allocating for a larger period of time. After having examined that data, it will

allocate resources for the next day. Correspondingly, in order to allocate resources for the third day it will use the data of the second day and so on.

The result of the prediction mechanism sets a binding upper limit which is the border for the next estimation of resource allocation by the Origin Servers. There is therefore a protection level against any malicious users who may try to buy large amount of resources from the beginning by exploiting the then lower price and then try to sell it in the Secondary Market during periods of high demand and low supply, aiming at super profit in the expense of clients in great need.

3. SECONDARY MARKET

The Secondary Market is the virtual space in which the resources can be bought and sold directly between the clients (Origin Servers) without the intervention of a supervising mechanism or the involvement of the CDN. It operates as a second chance place for origin servers to buy more resources at the time they need them to serve their needs. Their activation as wannabe buyers in the Secondary Market is a direct consequence of an initially (when they bought resources from CDN) bad estimation of their needs, having as a result the urgent need to buy more resources than the amount specified in their original agreement with the CDN. Simultaneously the Secondary Market serves also as the place where origin servers with surplus of resources have the option to sell a percentage of them in an attempt to gain back a part of the money they had initially spent on their purchase from the CDN, after it was determined that a partial amount of the bought resources was not necessary in the end.

The function of the Secondary Market Mechanism (SMM) is equivalent to that of a ledger where the seller records an entry in the ledger with a list of information about the stocks offered and in the meantime the buyer is looking for the cheapest stocks among the stocks offered.

When the -already initially purchased- resources of a client OS are underused, the client OS can sell a percentage of these resources through the SMM, usually at a lower price than that of the resources sold directly from the CDN. The difference between the selling price of the OS and the selling price of the CDN is aimed at enabling the seller OS to dispose of its excess of resources in the Secondary Market and to gain additional revenue by reducing its damage and thus making the most of the initially acquired resources. Of course, the price at which the OS offers to sell its excess of resources depends -besides on its knowledge of the fixed purchase price for the resources offered by the CDN- on the supply and demand relationship of the specific resources at that time in the market.

Respectively, when an OS's available resources are insufficient to meet its needs then the OS can turn to the Secondary Market to seek to buy the resources it needs by targeting a purchase price lower than that offered by the CDN to sell it the resources it requests. In the case where the CDN has available resources for sale, the client is very likely to find a much lower buying price if it is addressed to the secondary market, because its peers, namely the sellers OSs know the selling price of CDN and therefore seek to become more preferable than the CDN to the potential buyers by setting a lower selling price of their own resources.

The usefulness of the existence of the secondary market is further enhanced in the case where CDN does not have available resources for sale and therefore the only way for a client OS with an initial forecast failure to meet his needs is to turn to other seller OSs who -although they want to sell their surplus- at the same time they know that the resources to be sold are highly demanded and they do no longer compete with an upper sales price limit, which would have been the selling price of the CDN. In this scenario, the competition is intensified between OSs rather than between the OSs and the CDN and selling prices can reach higher levels than before. The buying cost could prove unbearable for the buyer OS in need, preventing it from covering its needs in a viable way.

In another case, the desirable quantity of resources may not be available in the Secondary Market even if a buyer OS is determined to pay any price. This state is considered as a 'miss' in the SM and it is another scenario of failure of the normal, smooth operation of the Secondary Market.

As a result of the aforementioned possibilities, a client should have a way of correcting his initial, failed forecast for the resource allocation; an additional failsafe, agreed at a time similar to his forecast, which -depending on the conditions prevailing on the market at the moment of the activation of the failsafe- gives the OS the opportunity to correct his initial assessment. This failsafe is called Stock Option.

Once we have introduced this new type of resource, the Stock Option, in the scope of this thesis we use this new failsafe to introduce and create a new version of the Secondary Market where instead of buying and selling stocks in this market the exchange taking place is that of Stock Options. This extra mechanism is placed on top of the existing secondary market of resources and serves as a third chance for the client OSs to correct their failed forecast before they have to turn to the CDN and buy resources from it with a penalized price.

4. STOCK OPTIONS

An additional mechanism is introduced on the scope of this thesis, playing a vital role in case of failure of an Origin Server to obtain the required resources from the Secondary Market and after having exercised all his available options. This proposed mechanism is an extra Secondary Market that offers its participants the ability to acquire the necessary resources in form of stock options. In this Secondary Market -instead of resources- the Origin Servers exchange stock options which can be exercised in order to purchase resources from the CDN in the present or in a later time and also in a lower buying price than the penalized price defined by the CDN. Taking into consideration the existing framework, it has now been enhanced with the proposed addition of this "Secondary Market of Stock Options". In the schema presented below (Fig. 7) the PRS model has been expanded with the additional functionality.

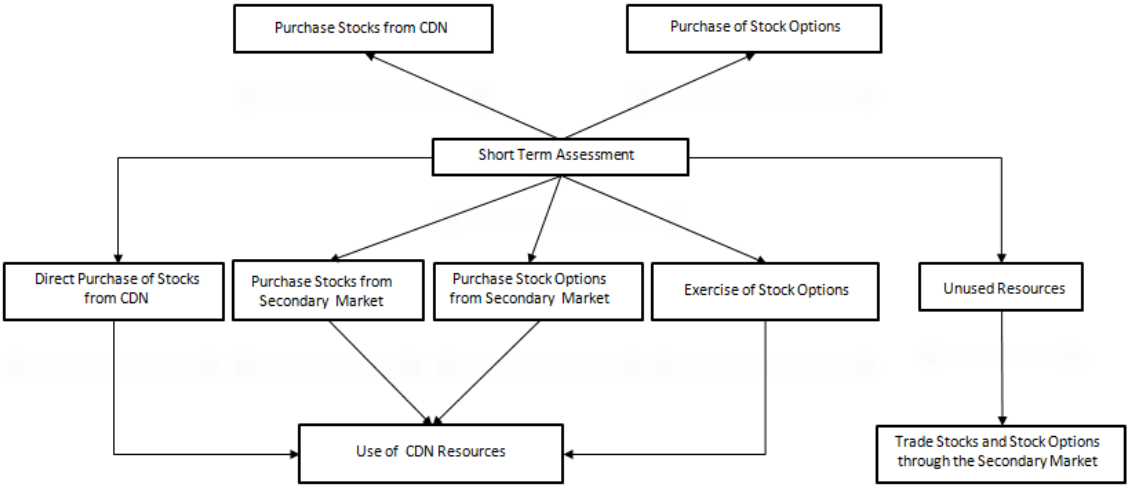

Figure 7: PRS Model including the Secondary Market of Stock Options

The usage of stock options in the proposed framework gives the clients of the CDN the ability to buy resources in case their prediction for the initial amount of resources they needed has been a "miss". Also they can buy these extra resources in a lower price than the penalized price they would have paid if they had bought them directly from the CDN. In addition, in case an Origin Server has an excess of stock options because of a "miss" in the prediction that caused it to buy stock options for a larger amount of resources that the one it eventually needed, this secondary market gives an Origin Server the option to sell these and gain back a part of the cost of the stock options that it initially bought, providing this way a more balancing financial environment.

In the following section we present what stock options are, the well-established formula that has been selected to calculate the price of the stock options, as well as the data that are used to feed the proposed framework in order to take into account the proposed new mechanism introduced in the PRS model.

4.1 Definition and Option Types

A stock option is a contract that provides its owner with the ability to buy or sell a resource in a predetermined price before or until a specific option expiration date. Stock options allow the buyer to do perform an action but not obligate him to do so. The buyer

exercises the option only if it is beneficial to him; otherwise he does not need to exercise it. Stock options are distinguished into two basic types; the Call Option and the Put Option [8].

- **Call Option (The right to Buy):** It's the most common option type and gives its owner the ability to buy an amount of resources (i.e. stocks) by paying a predetermined, fixed price during a specific time period until -or at- the specific option expiration date.

- **Put Option (The right to Sell)**: The Put Option is a stock option which can be considered the exact opposite of a Call Option. It gives its owner the ability to sell an amount of resources (i.e. stocks) in a specific predetermined price during a specific time period.

The time margin for the exercise of options varies depending on the expiration date of the contract. Stock options that can be exercised in each and every moment until the end of a specific date of a time period are called American Stock Options. On the other hand, stock options that can be exercised only at the date that their contract expires are defined as European Stock Options [8].

For the Call Options, it makes sense to exercise them when the current buying price of the resource is greater than the price of the stock option. For example, if a stock is worth 100€ and the stock option gives the buyer the ability to buy the stock at a strike price of 50€ then the gain of the buyer and the value of the stock option is 50€. In that case the buyer exercises the option. In the case that the current stock price is 40€, which means that it's lower than the stock option strike price, then the potential buyer will not exercise the stock option because he can buy the resources at a lower price. Therefore, the value of this call option is 0 (zero).

For the Put Options, it makes sense to exercise them as long as their owner can sell their shares at a price higher than the current value. For example, if the value of the stock is 50€, the owner of the put option can sell his stock at 60€ as dictated by the option; if he succeeds his gain -and also the value of this option- would be 50€. Therefore, the value of the stock option is increasing as long as the value of the stock decreases. In the case where the value of the stock is 70€ and the put option gives its owner the right to sell the stock at 60€, then the seller will not exercise the option and as a result, the value of the put option would be 0 (zero).

4.1.1 Structural Features of Stock Options

The fundamental, structural features and characteristics of Stock Options are summarized below [8].

1) **Subject Product:** The underlying product may be a title or share. In our case, the product is either the available bandwidth or disk capacity.

2) **Contract Size:** The size of the stock options includes the number of shares (in our case, the bandwidth and disk capacity) that will give the holder to sell / buy.

3) **Expiration Date:** It refers to the time frame within which a right can be exercised.

4) **Premium :** It is the monetary value to be paid by the buyer of the right

5) **Stock Option Right:** The right to buy (call option) or sell (put option) resources.

As it has been previously mentioned, the goal of expanding the model with the stock options mechanism is to be able to avoid penalized prices (i.e. higher prices than normally because of an extra penalty) for buying from the CDN in the event of a lack of resources that an OS may need while these need cannot be covered by hinging to the mechanism of the secondary market for extra resources. This fact, based on the aforementioned value options, concludes that the higher the penalized price for the buy of resources from CDN, the greater the stock option value for the purchase of resources from CDN will be.

Based on what has been covered so far, the goal of this thesis is to propose and present a way of costing the available CDN resources, which will enable CDN customers to pre-allocate these resources by paying a premium fee that will be evaluated based on the Black-Scholes model. The Black-Scholes model will be used to determine the purchase cost of the stock options that the CDN's customers can exercise in order to buy their unused CDN resources when they are needed at a later time. These stock options will act as the safeguard of the clients for acquiring the desirable amount of resources at the moment they need them and in the same time for avoiding the higher, penalized prices that would otherwise have to pay for the direct buy from the CDN at a later time. Also, this proposed model discourages economic blackmail by the exploitation of a situation, i.e.in which the customers would have been being the subjects of blackmailing by having to face excessive pecuniary requirements by other clients on the secondary market, at a market price much higher than the one guaranteed when buying a certain stock option.

4.2 Black-Scholes Model

The Black-Scholes model (also known as Black-Scholes-Merton) is a mathematical model for the evaluation of stock options. The equation, formulated in 1973 by economists Fisher Black, Myron Scholes and Robert Merton in their "The Pricing of Options and Corporate Liabilities" article and published in the Journal of Political Economy, is the first widespread royalty- option. Indeed, in 1997, they were honored by being awarded the Nobel Prize of Economics for their work and their contribution to the economic science.

The model calculates the theoretical value of stock options by taking into consideration as the inputs of the formula: the current price of stocks (resources), the dividends, the stock option exercise price, the expected interest, the time frame until the expiration date of the valid exercise period as well as the expected volatility which can be determined based on several factors.

The function can be used in the context of European-style stock options, i.e. in case where the exercise of a stock option can only take place on the expiration date of the stock option and not before then.

Overall, the standard version of the Black-Scholes model adopts the following assumptions:

- European-style stock options, which can only be exercised at the expiration date and not on any moment during the option's valid period.

- Dividends are not paid over the life duration of the stock options.

- The market changes in unpredictable way (its fluctuations cannot be forecasted).

- There are no additional costs when buying and selling stock options.

- The risk-free Rate of Return as well as the rate of Volatility of the stock price are known and constant values.

- The returns from the sale of the stocks are evenly distributed.

The equation is depicted in the following image:

$$C = SN(d_1) - N(d_2)Ke^{-rt}$$

$$d_1 = \frac{\ln\left(S/K\right) + \left(r + s^2/2\right)t}{s \cdot \sqrt{t}}$$

$$d_2 = d_1 - s \cdot \sqrt{t}$$

C = Call premium
S = Current stock price
t = Time until option exercise
K = Option striking price
r = Risk-free interest rate
N = Cumulative standard normal distribution
e = Exponential term

s = St. Deviation
ln = Natural Log

Figure 8: Black-Scholes Equation for Stock Options Pricing

The determination of the Call Option value for a certain resource (C) is the result of a function with a plethora of variables:

- *Current price of the resource* (**S**)

- *Time until the stock option expires* (**t**), which can be expressed as a fraction of one year.

- *Call Option price* (**K**)

- *Risk-Free Rate of Return (**r**)*: The Risk-free Rate of Return is the theoretical rate of an economic investment without any risk. The rate expresses the profit rate that the investor in a risk-free investment will receive after a period of time. In practice, this hypothetical value reflects the lowest profit rate that an investor/customer is targeting because -given the risk involved in any investment in the real economy- it is certainly required that the potential profit is greater than the profit of the risk-free rate.

- *Cumulative standard normal distribution* (***N***)

- *Standard Deviation of the distribution (***s)**

- *Natural log and exponential term* (**e**, **ln**)

The equation can be analyzed into two parts; the first part [S*N*(d1)] is the multiplication of the price with the expected change in market value in relation to the change in the underlying price. The product shows the expected benefit of a direct purchase. The second part of the equation defines the current value when exercising the Call Option at the maturity date, as it is dictated by the definition of European-style stock options. The market value is then calculated by the difference between the two parts of the equation.

As it has already being established the Black-Scholes Formula is used for the evaluation of European Stock Options where the buyer may exercise the option only at its expiration date as opposed to American Stock Options where the exercise can occur at any moment until the expiration date of the option. The ability to exercise an option at any given moment is extremely important and useful for a CDN's customer (in this case, an Origin Server) in the context of resource allocation because the moment of high demand of resources from the Origin Server's client is not known to the OS beforehand and can occur without being based on a calculated pattern. Therefore, as the goal is to have a balanced ecosystem of both CDNs and OSs with accurate allocation predictions and resources reservations in the long run, the customers of a CDN should be able to exercise their stock options for the buy of the requested amount of demanded resources at any moment in time.

In order to use the Black-Scholes formula in our model we will be examining a specific case of American-type Stock Options, where no dividends are paid. In our case, a dividend does not serve any purpose and by defining this, the use of the formula is permitted for the evaluation of an option.

More specifically, American-type Stock Options can be evaluated if they have been issued for stocks with no paid dividends because of a very important property: It is never useful to exercise an option before its expiration date. There are two reasons that lead to this conclusion:

a) First of all, just having an option for an action instead of the obligation to exercise it at a specific moment is by definition a safeguard for the owner of the option. For example, an unfavorable change in the share price would result in losses for the owner, but ownership of the right would secure the owner from these changes.

b) Secondly, there is the concept of the ‘time value’ or ‘value in time’ of money. Paying the price ‘A’ sooner rather than later means that the owner of the option loses a part of the ‘time value’ of the money, a part that could have been achieved in the remaining time of the right until it expires.

The property of “non-exercise” and the absence of paid dividends enable and American-type Stock Option to be eligible for evaluation by using the Black-Scholes formula [9].

In conclusion, it is obvious that the Black-Scholes model is a key tool in the hands of anyone who wants to determine the value of stock options. Any investor can use it to find out the value of a Call Option and whether buying is worth as an opportunity for investment. On the other hand, an investor may be prevented from a financial loss when the assessment based on the Black-Scholes model suggests that the value of an option is considered as potentially damaging should such an investment be made.

5. ORIGIN SERVERS EXERCISE SCENARIOS

Origin Servers can be triggered to participate in the market in multiple cases. This chapter covers the trigger points of origin servers and their potential actions in case they have failed in their initial prediction and subsequently do not possess the amount of resources they need in a specific day.

5.1 Buying Resources from the CDN by exercising Call Stock Options due to the lack of resources in the Secondary Market

Scenario 1: In this case the origin server will exercise its stock options due to a 'miss' in the secondary market. In this scenario the following considerations are taken into account:

- The requested amount of resources (i.e. bandwidth, disk space etc.) is not available for sale in the secondary market of resources.

- The resources' acquisition will be completely covered by the exercise of the call option initially bought by the OS. The CDN will sell the resources to the OS based on the terms agreed and described on the stock option that was exercised.

- The call options exercise for the buy of resources by the client OS takes place before the expiration of the stock options.

As an example the following situation of the secondary market is presented. Based on the table below, the resource type exchanged in the market is in this case bandwidth and there are three OS that have available bandwidth resources for sale in the secondary market with different market prices.

Table 1: Secondary Market Scenario 1

Secondary Market Resources Sales Board				
Os ID	**Num. Of Resources for Sale (in GB)**	**Availability (Expiration Date)**	**Market Price**	**Resource Type**
1	11	14/11/18 23:59	1.2 €	Bandwidth
2	10	21/11/18 23:59	1.4 €	Bandwidth
3	5	31/12/18 23:59	1.8 €	Bandwidth

Supposing that a client OS (ID 4) needs 30GB of bandwidth he will initially turn to the secondary market to buy resources. Once the client discovers that there no available resources in the secondary market, which is called a "miss", he will exercise his call option as it is dictated by the stock options contract in his possession and will eventually acquire the extra resources from the CDN in a lower price that the penalized one. By having bought stock options from the CDN, those SO give him the right to buy extra bandwidth of 30GB by paying an option strike price of 1.2€/GB instead of the penalized

price of 2€/GB that the CDN requires in case an OS wants to buy extra resources from the CDN on a later time because of an initial miss in the prediction on the client's part.

5.2 Buying Resources from the CDN by exercising Call Stock Options due to the Price difference compared to the Secondary Market

Scenario 2: In this case the stock options exercised will be triggered when the requested amount of resources is available in the secondary market but the market price is higher that the strike price of the call option an OS already possesses and it is not beneficial for the OS to buy resources from the secondary market. The following considerations are fundamental in this scenario:

- Resources are available in the secondary market for sale but the market price is higher than the strike price of the call option an OS already possesses.
- Resources will be completely covered by the buy from the CDN by exercising the call option.
- The call options exercise for the buy of resources by the client OS takes place before the expiration of the stock options.

As an example the following situation of the secondary market is presented where the resource type exchanged in the market is in this case both bandwidth and disk space. There are three OS that have available bandwidth resources for sale in the secondary market with different market prices and one OS that offers disk space for sale.

In case an OS (OS ID 4) has already bought stock options from the CDN that allow him to buy additional 20GB of disk space with a strike price of 0,5€/GB and additional 2GB/s of bandwidth with a strike price of 1€/Gbps instead of having to pay the market prices presented below (Table 2) or the CDN's penalized prices of 1€/GB and 2€/GB respectively.

Table 2: Secondary Market Scenario 2

Secondary Market Resources Sales Board				
Os ID	**Num. Of Resources for Sale**	**Availability (Expiration Date)**	**Market Price**	**Resource Type**
1	11	14/11/18 23:59	1.2 €	Bandwidth
2	10	21/11/18 23:59	1.4 €	Bandwidth
3	5	7/12/18 23:59	1.8 €	Bandwidth
5	25	7/12/18 23:59	0.75 €	Disk Space

Having checked on the secondary market first, the client OS observes that although the requested amount of resources is available there- the market prices are higher than the prices guaranteed by the stock option he has previously bought and paid for. This is also considered a miss since the client will prefer to exercise the call option in his possession and experience the value of his stock option by acquiring the resources he needs in a lower price that the market price and the penalized price of the CDN.

5.3 Combination of Resources from the CDN and the Secondary Market

Scenario 3: In this scenario the stock options mechanism (call option) will be activated when only a part of the requested amount of resources can be found in the secondary market and the rest will be bought by the CDN by calling the stock options. The following considerations are taken into account:

- A part of the requested amount of resources is available in the secondary market for sale based on the need of the client OS
- A part of the requested amount of resources will be acquired by the CDN by calling the stock options the OS has already bought and paid for.
- The resources available in the secondary market for sale have an equal or lesser value compared to the resources that can be acquired via stock options.
- The call option by the client OS is exercised before the expiration date of the stock options.

Table 3 presents a current state of the secondary market where both bandwidth and disk space are exchanged. A client OS possesses stock options that allow him to buy the necessary 20GB of extra disk space at a strike price of 0,5€/GB and 2Gbps of extra bandwidth with a strike price of 0,9€/Gbps instead of paying 1€/GB and 1,5GB€/Gbps respectively, namely the penalized prices of the CDN.

Table 3: Secondary Market Scenario 3

Secondary Market Resources Sales Board				
Os Id	**Num. Of Resources**	**Availability (Expiration Date)**	**Market Price**	**Resource Type**
1	11	14/11/18 23:59	1.2 €	Bandwidth
2	10	21/11/18 23:59	1.4 €	Bandwidth
3	5	7/12/18 23:59	1.8 €	Bandwidth
6	5	15/3/19 23:59	0.45 €	Disk Space

The search for the resources in the secondary market results in a “partial miss” because there are only 5 GB of disk space available for sale at a desired price. In such a case, the client OS will combine his options in order to cover his needs. The OS will buy those 5GB since the market price is lower that the strike price of his stock option and will exercise the call option in order to acquire the rest of the requested amount of disk space and bandwidth by the CDN at the strike price (and not the penalized price).

6. COST PLANNING AND PRICING POLICIES OF SERVICES

Having already established the significance of a stock options and following the trigger point possibilities examined in the previous chapter it is now clear that the exercise -or not- of a stock option is a critical decision which depends on a series of variables that have determined the state of the ecosystem (CDN, OS) and its mechanisms (Secondary Market) in a specific time period.

A fundamental variable in the determination of a stock option value is the strike price and more specifically its comparison to the (secondary) market price and the penalized price of the CDN.

The mapping of the pricing policy followed by CDN in today's era offers us the opportunity to evaluate the different approaches that can be recorded as scenarios - case studies for the evaluation and use of stock options.

In the context of the analysis of the service costing and pricing factor, it is necessary to define the basic principles and concepts governing service costing.

6.1 Principles and Definitions of establishing a Pricing Policy

Before analyzing the costing policies, it is necessary to look at fundamental concepts that shape the pricing policy of each service, particularly those offered by Content Distribution Networks. The main concepts are presented below:

- Billing Period

The billing period refers to the time period for which the customer is charged. For most services, the billing is calculated on a monthly basis, while rarely binary, biannual and annual billing periods are met.

- Minimum Contract Duration

Most contracts define a minimum duration period of one month but there are cases of large CDNs such as AKAMAI where the contract defines a minimum duration of 12 months.

- Seasonal Rebate Policies

In accordance to the minimum duration of the contract for a service there also seasonal rebate packages offered to the clients. For example, it is common for a CDN that offers service contract with a minimum duration of one month to offer an annual contract and charge the client only for 10 out of the 12 months. By providing a discount of 16,67% the CDN is aiming at the commitment of the customer for a larger period of time with a competitive price.

The use of CDNs for the allocation of resources is a direct result of the customer's need for resources. Since demand may be low over a certain period of time, as is the case for example in the low demand for live streaming of sports channels in the summer months where there is not much content like the rest of the year, it makes sense for the CDNs to seek to keep their customers for a longer period by offering them the appropriate incentives. The aforementioned example of the annual 2-month discount contract is a feature of this strategy.

- Minimum Seasonal Cost

Any pricing policy can offer a minimum seasonal cost as a fixed fee.

- Ability to share resources from multiple Origin Servers of common ownership

Resource sharing is a relatively recent option, according to which many Origin servers with a common owner can share the resources purchased by CDN based on their actual needs. Shared ownership is a prerequisite for sharing resources based on the needs of the owner and for optimizing the use of these OS while simultaneously minimizing unused, pre-allocated resources and therefore remedying failures of the prediction mechanism. An example of applying this feature is ‘Stackpath’, where there are the following sharing tiers: 5 Origin servers of common ownership can share 200 GB/month, 10 OS can share 800 GB/month, 50 OS can share 2 TB/month, 250 OS can share 7.5 TB/month and an unlimited number of OS can share up to 25 TB/month.

Additional information about the strategy and policies of one of the most dominant storage providers in the current market, namely Google and its Google Cloud Storage, is presented in Annex I.

7. PROPOSED MODEL: TRADING STOCK OPTIONS IN THE SECONDARY MARKET

In this chapter we analyze the proposed model for an efficient resource allocation environment between CDN and Origin Servers where stock options continue to play an integral role in resource management and an extra secondary market for stock options exchange between origin servers is introduced. Namely, we present the assumptions taken into account and the functionality of the model in regards to the stock options evaluation and pricing as well as how the CDN and the Origin Servers are going to interact with each other and how the process of resource allocation takes place.

One of the initial tools used in the model is the Black-Scholes formula for the stock options evaluation and the determination of their price. As it has been previously described (in chapter 5) the Black-Scholes equation is used for the evaluation of European-type stock options but in our model the assumptions have been made (i.e. no dividends paid) in order for American-type stock options to be evaluated. American-type stock options are perfect for our proposed model in terms of modeling the resource allocation schema between CDN and Origin Servers using stock options since it is not known beforehand the time that an origin server will require extra resources. Therefore the customers (origin servers) of the CDN must be able to exercise their stock options anytime as long as the options are valid.

7.1 Stock Options Mechanism Parameters

As it has been previously mentioned (see Chapter 5) in order to determine the value of a stock option there is a plethora of parameters that need to be taken into account. In the proposed model of this thesis, these parameters are determined as indicated below:

R: Total available resources of the client (OS)

Each client (OS) potentially possesses **R** units of the resource he is interested in, whether it is bandwidth, disk space, storage space etc. This variable represents the sum of the total amount of resources that is directly available to the client and can already be used for its needs plus the quantity it will obtain from the CDN through the exercise of the stock options but also plus the free resources available for purchase from the CDN, aside from any purchase contract.

More specifically:

R = Client's current resources + Resources by calling a stock option + Resources that can be bought from the CDN separately

x: Number of resource units occupied at any point in time either for serving the client's current needs or as pre-allocated resources for future use by the client.

t: The stock option' s expiration time, assuming it has been issued when t=0.

S: The current market price of the resource as it is being sold by the CDN to its clients.

b: The base unit of the resource. In our case if the resource is bandwidth it has the value of 1GB/s and if it is storage space it is 1GB.

B: The total amount of the resource that the client wants to pre-allocate.

v: The amount of resource units requested by the client (B = v*b)

τ: The amount of time for which the client pre-allocates its resources. It represents the period of the validity of a stock option, from start to finish.

E: The price at which the client is able to buy the resource by exercising the call option, namely the right to purchase at this price using a pre-owned stock option. This price is called 'strike price' and is inevitably compared to the 'market price' of a resource.

C: The cost of the client to obtain the stock option (the premium).

7.2 Stock Options Mechanism Analysis

The proposed model is analyzed into a mechanism of discrete phases. The algorithm and its implementation into simulations of different scenarios consist of three cornerstones:

- the Data Preparation phase on the beginning of the simulation,
- the repeatable Daily Operation process for each day of the simulation and
- the Data Conclusion phase where the results of the simulations are gathered after the simulation has ended.

Based on the above, these cornerstones are further analyzed in the following subchapters.

7.2.1 Data Preparation

In the beginning of the proposed stock options mechanism -and before the pre-allocation of resources and stock options take place- the CDN performs an evaluation of the stock options as well as their determines their pricing. This is performed taking into consideration the following steps:

1) First, we should consider which cost pricing model/policy will be used by the CDN in order to determine (as an input parameter in the mechanism) the minimum acceptable price per resource (bandwidth/disk space) that an OS client has to pay for the allocation of the resources he wants.
2) The CDN, having calculated the resource's price per unit, then calculates the value of a stock option that expires at a later time t + t, using the Black-Scholes equation:

$$C = SN(d_1) - Ee^{-r\tau}N(d_2)$$

Where $d_1 = \frac{\left[\ln\left(\frac{S}{E}\right)\right] + \left(r + \frac{\sigma^2}{2}\right)\tau}{\sqrt{\tau\sigma^2}}$

$$d_2 = d_1 - \sqrt{\tau\sigma^2}$$

The value of the stock option at time **t** is **C** (the value of this parameter is instantaneous) and depends only on the following parameters:

S: The value of a resource unit at time t.

E: The strike price, namely the exercise price of the option. It is essentially the price that CDN evaluates that it will negotiate its resources at the time of the expiration of the option.

r: The risk free rate of return per unit of time. Based on historical data, having purchased stock options in the past and whether or not it eventually needed these resources, the client has already formulated a percentage of utility without risk for its investment in stock options.

σ^2 : Variation (per unit of time (i.e. year)) of the unit cost of the resource.

τ : The time period until the expiration of the stock option.

N(d) : The probability that a normal random variable is less than or equal to d.

An important question that is raised by now is how the CDN will determine the price of the option. In our model this variable will be determined based on the cost model of the CDN based on which the low, normal and high prices are determined for each resource plan that the CDN provides. The following table depicts the cost plans based on the total volume of the resource along with the normal cost, low penalty and high penalty prices of these resources per cost plan.

Table 4: Cost and Total Volume Details of CDN

Cost Plan	TV (GB)	Normal Cost	Low Penalty	High Penalty
P1	0-10	0.035	0.033	0.042
P2	10-100	0.03	0.028	0.036
P3	100-1000	0.025	0.023	0.03
P4	1000-10000	0.02	0.019	0.024
P5	10000-100000	0.015	0.014	0.018
P6	100000-1000000	0.012	0.011	0.014

Based on the above table; for our model we have chosen to take into consideration the low penalty price that the CDN provides, given that all players in the ecosystem aim to cooperate in a socially optimal and not in a greedy way. In that direction, also in the case where an OS has a surplus of resources and/or stock options to sell, there will be no extra fees to gain as an exploit of another client in need.

In addition to the strike price parameter (E), also the exercise period (τ) of the stock options must be determined. For our model, across all implemented scenarios, we have established that the maximum time period that any stock option bought by the CDN can be exercised will be one month (30 days). For that reason, as a base for all examined scenarios, a reference table is created and “published by the CDN” at the start of the simulation. Taking into account the Black-Scholes formula, it depicts all possible prices of a stock option according to the time until its expiration and for each one of the 6 cost plans of Table 4. Origin servers can use the data on this table in order to determine how much money they will have to spend on the purchase of stock options. This reference pricing table can be found in Annex III.

7.2.2 Daily Operations Process

1) Once the day has begun, each Origin Server makes its prediction about the amount of resources it will need for the current day, taking into consideration its previous days’ usage.

2) Next, each Origin Server will perform an evaluation of the resources it has and will determine how many resources will need to obtain in the form of stock options from the CDN. This amount of resources in form of stock options, described in the code as “Predicted SO”, is expressed as a percentage of the predicted amount of resources and is one of the simulation’s input arguments.

3) At the time when the Origin Server buys from the CDN the resources based on the predicted traffic for the day, also has the following options in terms of a stock options buy:
   - To proceed with the pre-allocation of resources in the form of stock options with each stock option valued at a price C, based on the volume of resources it wants and their time until expiration, as it has been published in the CDN’s pricing catalogue. If this is case, then the Origin Server buys these stock options and these underlying resources will be marked as allocated from CDN’s side in order not to be taken into consideration for requests from the rest of the origin servers.
   - Not to proceed with pre-allocation of resources in the form of stock options, if it has determined that it has enough resources for the day and/or stock options not exercised yet from previous days.

4) As the day progresses, the servers come across their real traffic, namely the actual demand of resources from the OS’ clients. In the next phase during the day, if the Origin Servers were not able to cover their needs with the resources they had initially bought, they go to the secondary market of resources, where all Origin Servers sell their excess in resources they had bought from the CDN in the beginning of the day. In this SM, the price that the resources are sold and bought is going to be lower or equal than the high penalty price that CDN has for the resources, if the Origin Servers were to buy them from CDN.

5) In the case where the activity of the Origin Server (potential buyer) in the SM of resources was unsatisfying due to lack of availability of the requested amount of resources and the possessed resources are still not enough for the real experienced traffic of the OS in time th , we distinguish the following cases:
   - If $t_h \leq \tau + t$, then the Origin Server will proceed with exercising its stock options and will provide to CDN the amount E=S(t_1).
   - If $t_h > \tau + t$, then the Origin Server will not proceed with the exercise of the stock options and will have lost the original amount C which had provided to CDN.

6) Once the phase of the stock options exercise has ended, the Origin Servers that were not able to fulfill their needs will search in the SM of stock options, where Origin Servers exchange their stock options of underlying resources.

7) In the case that the Origin Servers still haven't covered their needs they will go to CDN to buy the remaining resources they need with the penalized price.

7.2.3 Data Conclusion

At the end of the simulated period the following daily and collective data has been gathered:

- CDN Data
  - Resources Sold
    - per day
    - per buyer OS
  - Stock Options Sold
    - per day
    - per buyer OS
  - CDN Wallet
    - Total
    - According to the selling price (normal/penalized)
- Origin Server Data
  - Initial Traffic and Stock Option Prediction
  - Real Traffic
  - Secondary Market of Resources Activity
    - action (buyer/seller)
    - Resources Exchanged
  - Secondary Market of Stock Options Activity
    - action (buyer/seller)
    - Resources Exchanged
  - Wallet
- Stock Option Data
  - record of all stock options
    - based on their state: A list of all SO, whether they were initially bought and in the end exercised, exchanged or even still valid at the end of the simulation.
    - based on their cost
    - by purchase day [1,…,364]
    - by buyer
- Collective Metrics about the scenario
  - Amount of times the origin servers had to buy resources by paying a penalty price.
  - Amount of Unused Stock Options
  - Cost of Unused Stock Options
  - Total Resource Expenses (for all OS)
  - Total Stock Options Expenses (for all OS)
  - 'SM of SO' Wallet: total monetary exchanges in the Secondary Market

The analysis of the data is presented in chapter 8.

In the Annex there are references to the code created to implement the aforementioned algorithm.

7.3 The Secondary Market of Stock Options

The main novelty introduced in the thesis is the introduction of an extra Secondary Market where the origin servers have the ability to exchange stock options of underlying resources. This SM takes place after the SM of exchanged resources and has the same concept: the Origin Servers are divided into Buyers and Sellers and aim to fulfill their needs either by selling their SO and increasing their Wallet size (Buyers) or by buying SO in order to acquire the underlying resources in an affordable price and be able to respond to their daily demand of resources despite the initial miss of their predicted traffic (Sellers). As the metrics analysis of the following chapter will show, the implementation of an extra secondary market proves quite useful in the direction of a balanced market with the following characteristics:

- Usefulness for Origin Servers that have previously bought Stock Options they do not need to exercise in the end, as they are now able to sell them to those in need and regain a part of their expenses.
- Minimization of the amount the Origin Servers have to turn to the CDN for the immediate of buy extra resources and pay the penalized price for them.
- Avoidance of exploitation of market players by the CDN.
- Balance between the CDN’s profit and the finances of the Origin Servers.

8. SIMULATION – SCENARIOS, METRICS AND RESULTS

8.1 Parameters: Constants and Variables

The constant characteristics for all simulated scenarios are:

- The underlying resource adopted for the simulation is Bandwidth (**BW**).
- All stock options metrics' values refer to amount of the underlying resource that the stock option represents. For example, the "Predicted SO" metric – that will be presented in subchapter 9.3 and expresses the amount of stock options an origin server estimates will need for the day- actually expresses the amount of resources that can be bought via exercised stock options. A value of 1000 "Predicted SO" means that the origin server estimates that will need stock options that, when called, 1000 units of resources will be bought at the strike price of the SO.
- Simulation Cycle Reference Period: **364 days**.
- Cost Plan Method: **Pre-Costed Plans**. There are tiers depending on the traffic volume. The more resources the OS buys, the less per unit they cost.
- CDN Penalty Policy: **Low Penalty price**. The price used for the calculation of the stock options by the CDN is based on the low penalty price of the respective cost plan from CDN.
- The ecosystem operates on a **socially optimal** notion of resource distribution and not greedy or antagonistically between players (Origin Servers and CDN).
- Reuse of unused resources: Origin Servers can reuse resources they had left from previous days. Resources can be transferred to the next day and the same applies for the stock options bought by the origin servers. Besides, the period of time realistically corresponds to the actual duration for use of resources by companies in the telecommunications market and CDN providers.
- Stock Options can be exercised at every day until the expiration date and not only at the day of their expiration. The Black-Scholes formula is used for the stock option price calculation as the stock options do not pay dividends (see chapter 5).

The examined scenarios have been determined based on the different values of the input parameters of the model.

- Existence of the Secondary Market of Stock Options: **True / False**
- Maximum Stock Options Duration (in days): **7 / 14 / 21 / 28**
- Percentage of SO bought at the start of the day based on the initially predicted traffic: **5%-10%-15%**

8.2 Exercising Scenarios

In total, twelve (12) scenarios have been chosen to be simulated with the proposed framework. In the next sections of the chapter are described: the input parameters of each scenario, the assumptions taken into consideration as well as the metrics which have been chosen in order to estimate the effectiveness of the proposed model.

The below table presents the number of scenarios tested along with the main input parameters for the simulation. The parameters were mentioned in 8.1 and have the following codenames:

- **SO Exchange in SM:** This parameter defines if the secondary market of stock options will exist or not during the simulation.

- **Max SO Duration:** This parameter defines the validity period of the stock options to be exercised during the simulation.

- **Percentage of SO:** The percentage of SO to be bought signifies the target percentage of stock options that the origin servers will hold each day of the simulation, based on the predicted resources that the they have bought at the start of each day.

Table 5: Exercise Scenarios

Scenario	SO Exchange in SM	Max SO Duration (days)	Percentage of SO
1	FALSE	7	10%
2	FALSE	14	10%
3	FALSE	21	10%
4	FALSE	28	10%
5	TRUE	7	10%
6	TRUE	14	10%
7	TRUE	21	10%
8	TRUE	28	10%
9	FALSE	6	15%
10	TRUE	6	15%
11	FALSE	14	5%
12	TRUE	14	5%

For the stock options, the following parameters have been configured for all the scenarios presented:

Table 6: Fixed Input Parameters of the Simulation

Parameter	Value	Description
Risk free rate	0.05	The rate of return on the riskless asset.
Volatility	0.01	The volatility of returns of the underlying asset.
CDN SO duration	30	This value signifies the maximum duration in days for which the CDN will have calculated the stock option's value.
Days of the simulation	364	The value signifies the number of days for which the simulation will run.

8.3 Metrics

The following metrics have been taken into consideration to compare the effectiveness of the framework based on the parameters set for each scenario. The metrics are

divided into the metrics that are specific for the origin servers and metrics specific to the CDN.

Origin Server Metrics:

- **Predicted SO**: The total number of stock options predicted to be needed by the Origin Servers during the simulation at the start of each day.
- **Predicted BW:** The total number of bandwidth predicted to be needed by the Origin Servers during the simulation at the start of each day.
- **Number of resources bought with penalty:** This metric shows the number of resources that where bought with penalty price from CDN during the simulation.
- **Times bought with penalty:** This metric shows how many times an Origin Server needed to buy from CDN with penalty price.
- **Unused SO Cost:** This metric shows the cost of the stock options that the Origin Server was not able to exercise before their expiration day.
- **Traffic (bandwidth) Expenses:** This metric shows the total expenses in bandwidth of an origin server during the simulation. It includes the expenses of buying the resources at the beginning of the day and the expenses of buying the resources from CDN with penalty price.
- **SO Expenses:** This metric shows the total expenses of in stock options of an origin server during the simulation.
- **Wallet SM:** This metric shows the wallet status of an origin server at the end of each simulation.

Content Delivery Network metrics:

- **Total Income:** This metric shows the total income of the CDN at the end of the simulation including the bandwidth and the stock option bought from the Origin Servers.
- **Income w/o penalty:** This metric shows the total income of the CDN from the resources (bandwidth) bought from the Origin Servers at the beginning of each day of the simulation not in the penalty price including the income from the sell of stock options from the origin servers.
- **Income with penalty:** This metric shows the total income of the CDN from the resources (bandwidth) bought from the Origin Servers in the penalized price at the end of each day of the simulation.
- **Income from SO:** This metric shows the total income of the CDN from the selling of stock options only during the simulation.
- **Income from BW:** This metrics shows the total income of the CDN from the resources (bandwidth) bought from the Origin Servers; either at the begging of the day or with penalty price at the end of each day of the simulation.

8.4 Results

The collective results of all aforementioned metrics and their comparative analysis are presented below.

Predicted Stock Options per scenario for each Origin Server

Table 7: Predicted Stock Options per scenario for each OS

	Origin Servers						
Scenario ID	1	2	3	4	5	6	Total
1	3,495,548,458	479,657,513	95,143,010	1,885,314	525,735,997	12,734,251	**4,610,704,543**
2	4,025,102,327	605,345,423	135,742,555	2,673,483	612,972,076	17,052,846	**5,398,888,710**
3	4,034,782,809	605,513,764	135,673,390	2,644,776	614,018,735	16,884,627	**5,409,518,101**
4	4,030,505,217	606,000,200	135,673,390	2,644,776	614,946,994	16,884,627	**5,406,655,204**
5	5,194,402,025	781,521,558	193,947,050	4,055,962	744,256,708	22,749,838	**6,940,933,141**
6	5,168,314,720	783,816,479	190,690,151	4,004,614	746,540,640	22,459,441	**6,915,826,045**
7	5,149,983,426	784,999,946	191,854,808	3,995,638	757,499,144	22,459,441	**6,910,792,403**
8	5,169,248,861	779,439,857	191,840,226	3,995,638	744,668,497	22,459,440	**6,911,652,519**
9	5,751,186,699	921,065,991	215,893,610	4,118,373	966,906,515	27,391,873	**7,886,563,061**
10	6,722,495,142	1,006,102,236	258,944,856	5,199,513	1,006,733,030	29,560,415	**9,029,035,192**
11	2,856,790,747	470,193,550	123,878,179	2,509,989	448,909,981	15,224,611	**3,917,507,057**
12	3,617,495,341	524,747,452	139,257,925	2,913,531	452,575,187	15,693,172	**4,752,682,608**

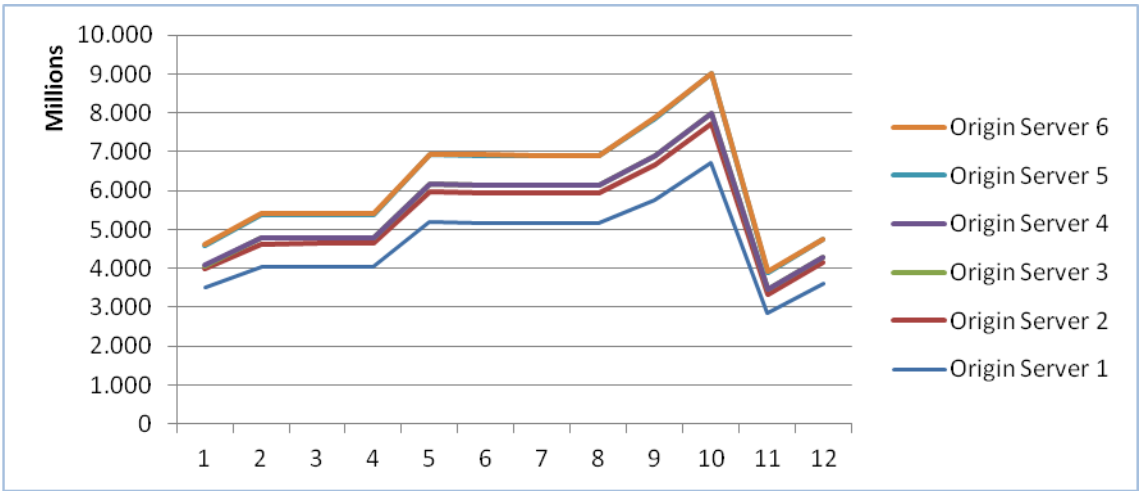

Figure 9: **Predicted Stock Options for each OS per scenario**

The above line chart shows that the fluctuation of the predicted stock options is always proportionate for each Origin Server in each scenario. Whether an increase or a decrease on the stock options prediction from the scenario to scenario, it is distributed evenly among all clients of the CDN.

The total amount of stock options, displayed on table 7, shows that the optimal scenario is no.11 where 3,917,507,057 stock options are expected to be needed for all servers during the year. This is quite interesting because in this scenario there is no secondary market of stock options. The lack of a SO exchange market means that there is no transfer of an option's ownership and due to the lack of reusability, new options will always be purchased when necessary. Despite that significant factor, the major

parameter that influences the low total amount of predicted stock options is the longer duration of stock options' life. Having 14 days for the exercise of an option, the origin servers can keep their options for a whole 2-week period in order to remedy in a day –or several days- of false predictions. Also, an important factor is that the origin servers acquire only 5% of their predicted traffic resources in stock options at the start of each day. They are not committed to buying a larger amount of options and possibly watching them remain unused until their expiration due to the fact that they have other previously bought options that can be used because they remain valid.

On the other hand the worst scenario is the no.10 where the combination of the input parameters result in a large amount of 9,029,035,192 predicted stock options. In this case, the existence of the secondary market of stock options combined with the short duration of 6 days until their expiration result in great volatility of the ownership of short-lived options.

Predicted Traffic per scenario for each Origin Server

Table 8: Predicted Traffic per scenario for each OS

	Origin Servers						
Scenario ID	1	2	3	4	5	6	Total
1	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
2	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
3	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
4	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
5	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
6	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
7	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
8	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
9	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
10	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
11	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**
12	114,712,531,054	18,405,690,133	4,223,553,000	88,696,772	15,190,369,712	478,104,952	**153,098,945,623**

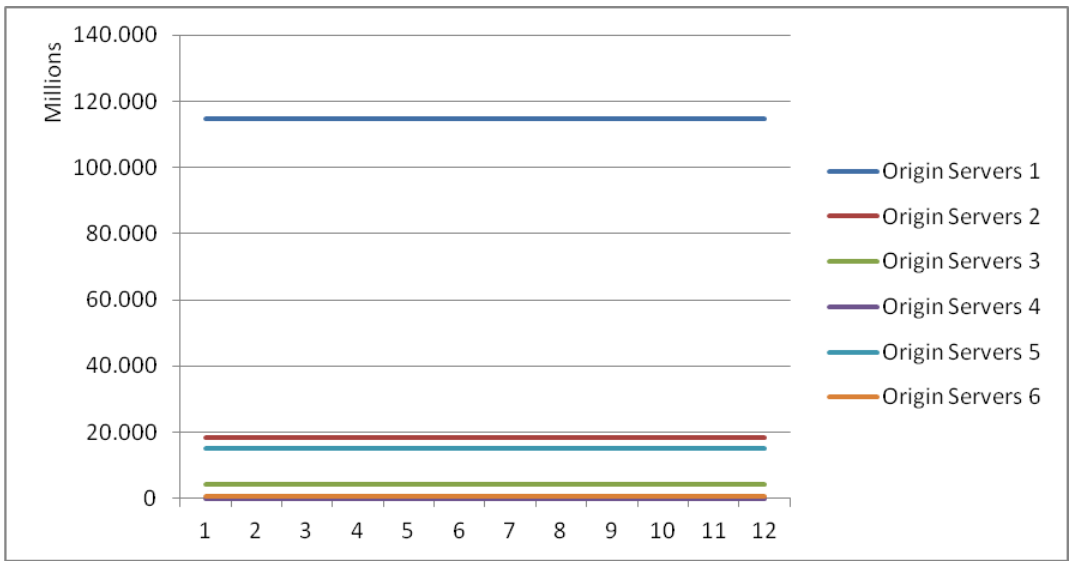

Figure 10: Predicted Traffic (Bandwidth) for each OS per scenario

The predicted traffic (i.e. resources in bandwidth) is a metric unaltered by the different parameters of each scenario because it solely depends on the prediction mechanism and is not influenced by the actions afterwards.

Times Bought with Penalty per scenario for each Origin Server

The number of times an Origin Server buys resources by paying a penalized price to the CDN is considered the most important metric of our simulation. The main goal of our proposed framework has been the minimization of the times that the Origin Servers will need resources from CDN at the end of the day and in this way will avoid additional wallet expenses by paying the penalized price specified by CDN.

Table 9: Times bought with penalty per scenario for each OS

	Origin Servers						
Scenario ID	1	2	3	4	5	6	Total
1	68	77	53	32	79	56	**365**
2	65	74	51	32	78	55	**355**
3	65	74	50	30	78	55	**352**
4	65	74	50	29	77	55	**350**
5	66	62	43	26	57	42	**296**
6	67	62	42	26	59	41	**297**
7	67	61	41	26	58	41	**294**
8	67	61	41	26	58	41	**294**
9	57	71	44	25	73	45	**315**
10	54	49	29	15	46	31	**224**
11	92	83	64	46	86	67	**438**
12	90	73	55	45	77	52	**392**

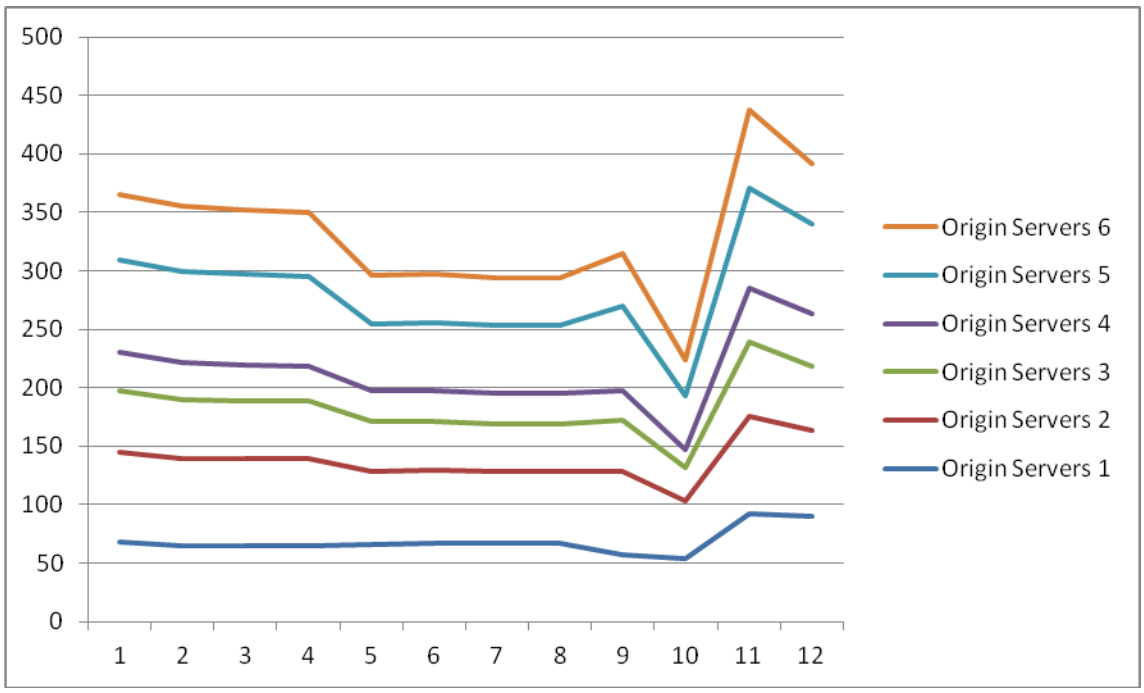

Figure 11: Times bought with penalty for each OS per scenario

Based on the table and diagram above and by the comparison of scenarios that, although have the same input parameters differ on whether or not the secondary market of stock options exists, we have determined that the existence of the extra SM has significantly reduced the times the Origin Servers needed to turn to the CDN for extra purchase of resources.

The following table displays the comparison of each duo of contradictive scenarios.

Table 10: Comparison matrix between scenarios

Comparison ID	(A) Scenario ID	Times Bought with Penalty in (A)	(B) Scenario ID	Times Bought with Penalty in (B)	Difference between (B) and (A)	Difference (%) between (B) and (A)
1	1	**365**	5	**296**	-69	**-18.90**
2	2	**355**	6	**297**	-58	**-18.63**
3	3	**352**	7	**294**	-58	**-16.48**
4	4	**350**	8	**294**	-56	**-16.00**
5	9	**315**	10	**224**	-91	**-28.89**
6	11	**438**	12	**392**	-46	**-10.50**

In each compared duo of scenarios, the scenarios of column '(A) Scenario ID' do not involve the extra SM where those of '(B) Scenario ID' do simulate this functionality. Apart from this difference, each duo has the same input parameters. It is determined that in every comparison, the (B) scenarios show a reduction in the amount of times bought with penalty due to the fact that the stock options exchange between the servers serves a lot of times as a way of avoiding a direct buy from the CDN. The biggest improvement is observed in Comparison #5 between scenarios #9 and #10. Despite the

fact that scenario #9 already has the lowest value among all simulated scenarios that do not involve the Secondary Market of Stock Options, it is remarkable that scenario #10 – by implementing the extra SM- presents an ever greater improvement by managing almost 29% fewer visits to the CDN.

On average, across all examined simulations, the number of buys with the penalty price has dropped up by 18.23 %. It is therefore concluded that the introduction of an extra secondary market improves this aspect of market operations by a significant percent.

Resources Bought with Penalty per scenario for each Origin Server

Table 11: Resources bought with penalty per scenario for each OS

	Origin Servers						
Scenario ID	1	2	3	4	5	6	Total
1	6,075,409,755	2,501,236,659	177,497,702	4,657,500	2,267,978,803	22,256,130	**11,049,036,549**
2	6,041,870,337	2,480,290,201	173,259,729	4,581,246	2,227,209,246	21,692,787	**10,948,903,546**
3	6,024,032,527	2,469,644,878	169,943,739	4,516,734	2,222,478,602	21,340,408	**10,911,956,888**
4	6,024,032,527	2,470,281,675	169,943,739	4,506,486	2,219,073,115	21,147,534	**10,908,985,076**
5	5,663,681,431	2,105,534,869	87,724,098	2,164,488	1,455,592,033	17,420,798	**9,332,117,717**
6	5,739,050,328	2,026,220,568	85,767,301	1,969,256	1,513,018,672	16,659,041	**9,382,685,166**
7	5,918,599,970	1,920,543,946	80,001,257	1,964,558	1,499,081,925	16,617,608	**9,436,809,264**
8	5,902,121,571	1,918,741,169	80,016,278	1,964,381	1,512,013,616	16,619,435	**9,431,476,450**
9	5,207,899,486	2,339,235,408	151,248,280	4,352,119	2,106,361,724	19,185,695	**9,828,282,712**
10	4,915,179,971	1,671,599,276	46,497,542	1,239,649	1,178,079,387	13,940,869	**7,826,536,694**
11	7,300,602,850	2,659,543,582	202,120,779	5,039,151	2,452,890,856	26,045,425	**12,646,242,643**
12	6,835,139,736	2,254,439,003	128,142,270	3,259,935	1,994,715,217	22,728,525	**11,238,424,686**

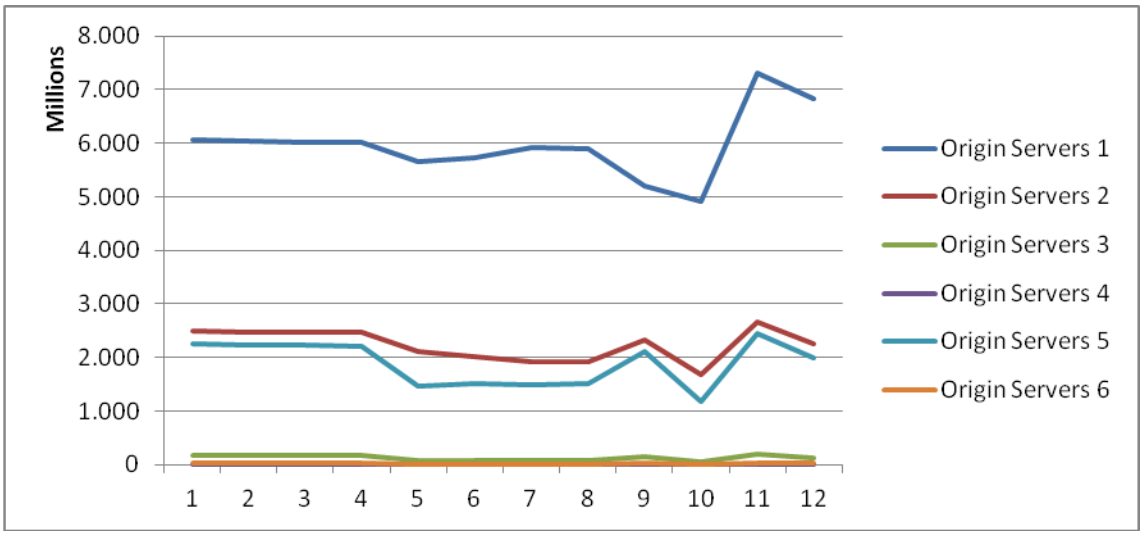

Figure 12: Resources bought with penalty for each OS per scenario

Continuing from the amount of times a server had to buy extra resources from the CDN with penalty, it is also necessary to review the amount of these resources bought, since these costs affect the server's wallet greatly.

The first general observation is that the scenarios which simulate the secondary market of stock options (ID={5,6,7,8,10,12}) present a lower total amount of 56,648,049,977 resources bought with penalty that that of 66,293,407,414 in the scenarios without the existence of the extra SM (ID={1,2,3,4,9,11}), with an overall average decrease by 14.55%.

Furthermore, in accordance to the previous presented metric, scenario #10 presents the lowest amount of resources bought with penalty with a decrease of 20.37% compared to the scenario #9 that does not implement the extra SM. This observation confirms the significance of buying a larger percentage (15%) of the predicted traffic in stock options while also buying them at a lower cost since the options have a shorter expiration date (6 days) that still covers the clients' needs. It also worth mentioning that the overall decrease reflects a decrease in the resources of all Origin Servers but not evenly distributed. Origin Server #1, which always has the largest amount of resources, experiences a smaller decrease of 6% compared to the other 5 players of the market.

Unused SO Cost per scenario for each Origin Server

An important element for determining the usefulness of stock options in the area of content delivery networks is the cost of unused options The origin servers seek to pre-allocate resources by buying stock options but if they ultimately do not exercise them, then the buying cost of unused options simply affects their wallet in a negative way. Also, their initial prediction was proven to be wrong.

This metric's importance is highlighted even more in the case of the existence of a secondary market, in which the options can be exchanged, offering an OS the opportunity to sell options that thinks might not be useful to its needs anymore. Instead of letting it expire, the options can be sold to other OS in need and the seller can regain some of its initial buying cost, depending on the current price of the option.

Table 12: Unused SO Cost per scenario for each OS

	Origin Servers						
Scenario ID	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
1	779,506	204,894	63,278	1,574	189,312	7,142	**1,245,705.86**
2	102,766	50,145	18,458	36,777	36,777	2,561	**211,333**
3	42,922	5,717	0	18,964	18,964	404	**68,141**
4	0	5,096	0	7,173	7,173	0	**12,303**
5	561,355	119,568	23,618	101,732	101,732	2,728	**809,516**
6	35,136	25,116	3,072	4,491	4,491	292	**68,157**
7	0	2,414	0	0	0	0	**2,414**
8	0	0	0	0	0	0	**0**
9	1,360,217	406,640	114,720	437,154	437,154	15,016	**2,336,699**
10	942,666	235,731	55,777	215,478	215,478	6,776	**1,457,444**
11	83,539	16,803	7,417	17,620	17,620	1,244	**126,885**
12	31,713	14,012	0	2,271	2,271	142	**48,227**

The results presented on table above show that each scenario of the extra SM presents a significant decrease compared to its respective scenario without the extra SM. Taking all 12 scenarios into account, the overall decrease of unused stock options cost reaches

an average of 40.37% when the secondary market of stock options comes into play (2,385,758 compared to 4,001,066.86 wallet units).

A further observation is that scenarios with higher costs of unused SO are those that simulate SO with a shorter expiration date. It can be explained since unused options of short expiration date have a lower usability to the potential buyer and should no one need them during that shorter timeframe, they present a higher likelihood of remaining unused until their expiration. It is also worth noting that, even when the secondary market of stock options exists, such options will be sold at a lower price compared to the ones that offer the potential buyer a larger duration of exercise.

Focusing on the individual Origin Servers, it is observed that there are cases where the servers have zero unused SO cost. Specifically, in scenarios, #3,#4,#7 and #8 there are origin servers which they have used all their stock options; either by exercising them or by selling their excess through the secondary market. In scenario #8 where the expiration time of stock options is 28 days we observe that all six Origin Servers have zero cost of unused stock options, effectively having experienced the optimal case of utilizing their bought options and/or regaining all cost invested in them.

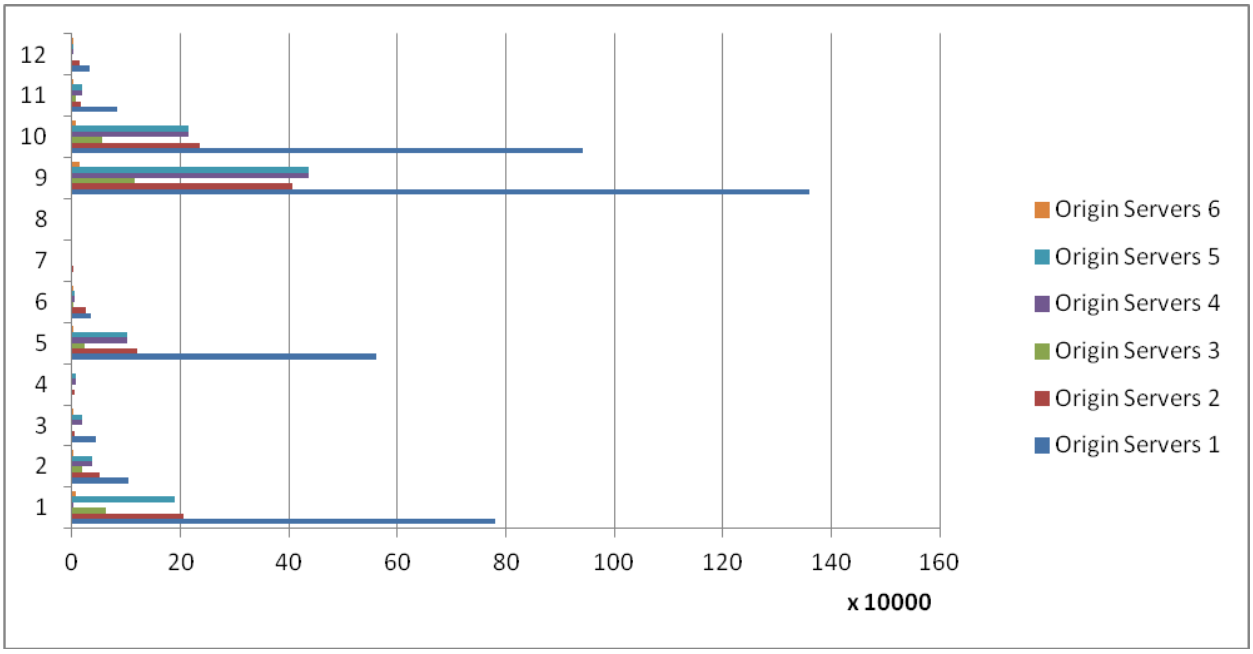

Figure 13: Unused SO cost for each OS per scenario

Unused SO per scenario for each Origin Server

In accordance to their cost, the amount of unused Stock Options is also an important factor of the model's success. The following table shows the amount of options that have expired without having been exercised, for each scenario and for each one of the Origin Servers. It needs to be mentioned that there are also stock options that remained valid until the end of the simulation, simply because these were bought during the last week of the simulation. Since the clients did not have enough days left to find out whether they would need to exercise them or not, such SO have been excluded from the table.

Table 13: Unused SO per scenario for each OS

	Origin Servers						
Scenario ID	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
1	517,334,884	101,263,958	31,216,393	774,850	93,565,671	3,516,174	**747,671,930.00**
2	92,564,129	24,510,071	8,987,755	303,712	17,981,553	1,241,105	**145,588,325**
3	40,700,244	2,745,634	0	64,214	9,178,724	192,874	**52,881,690**
4	0	2,418,383	0	15,928	3,435,138	0	**5,869,449**
5	342,253,110	59,106,786	11,651,972	253,735	50,279,688	1,342,839	**464,888,130**
6	33,018,272	12,279,712	1,495,816	23,947	2,186,509	141,602	**49,145,858**
7	0	1,160,101	0	0	0	0	**1,160,101**
8	0	0	0	0	0	0	**0**
9	1,070,243,995	201,353,814	56,703,524	1,456,988	216,478,266	7,408,485	**1,553,645,072**
10	744,177,771	116,719,930	27,570,760	501,197	106,700,446	3,342,581	**999,012,685**
11	40,865,479	8,200,520	3,608,354	127,104	8,596,688	602,887	**62,001,032**
12	15,491,964	6,854,896	1	43,173	1,105,864	68,797	**23,564,695**

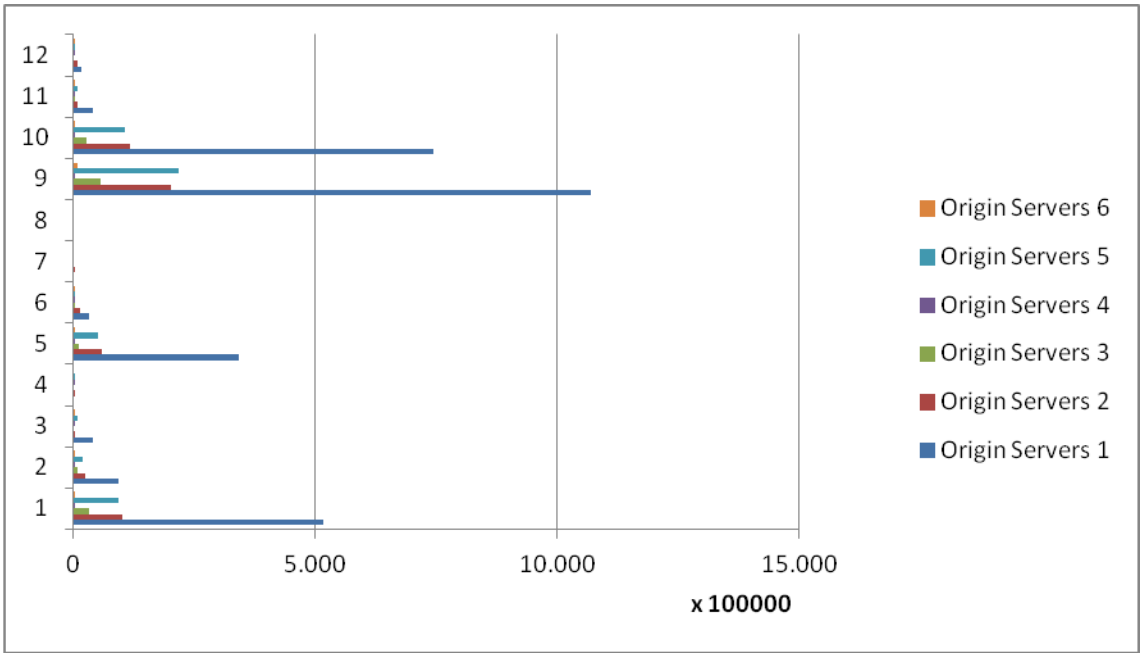

Figure 14: Unused SO for each OS per scenario

The table and the graph highlight several points. The scenarios implementing the extra secondary market present an overall decrease of unused Stock Options. Among them, scenario #8 that involves stock options of longer duration (28 days) than the other zeroes the amount of unused options for all six OS. The life expectancy of an option is inversely proportional to the probability of it remaining unused. Furthermore, it is noted that increasing the percentage of stock options bought (related to resources bought) results in similar increase of unused stock options.

Expenses in Bandwidth per scenario for each Origin Server

Expenses in bandwidth reflect the majority of a server's expenses, since resources are bought at least once per day (at the beginning). Bandwidth is also exchanged in the

secondary market of resources and it can also be bought directly from the CDN by paying the penalized price, if the client has not previously managed to cover its traffic needs.

Table 14: Expenses in Bandwidth per scenario for each OS

	Origin Servers						
Scenario ID	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
1	3,044,535,099	369,243,172	96,533,268	2,745,153	303,325,814	12,420,428	**3,828,802,933**
2	3,044,535,099	369,243,172	96,533,268	2,745,153	303,325,814	12,420,428	**3,828,802,933**
3	3,044,535,099	369,243,172	96,533,268	2,745,153	303,325,814	12,420,428	**3,828,802,933**
4	3,044,535,099	369,243,172	96,533,268	2,745,153	303,325,814	12,420,428	**3,828,802,933**
5	3,107,459,745	386,093,860	97,842,981	2,766,858	298,249,562	12,490,438	**3,904,903,444**
6	3,112,322,486	386,145,111	97,842,981	2,766,858	299,831,490	12,490,438	**3,911,399,364**
7	3,113,202,545	378,947,732	97,842,981	2,766,858	302,364,259	12,502,338	**3,907,626,713**
8	3,118,879,611	379,456,362	97,842,981	2,766,858	303,215,609	12,502,338	**3,914,663,759**
9	3,044,535,099	369,243,172	96,533,268	2,745,153	303,325,814	12,420,428	**3,828,802,933**
10	3,056,336,377	379,492,977	97,842,981	2,766,858	291,295,601	12,502,338	**3,840,237,132**
11	3,044,535,099	369,243,172	96,533,268	2,745,153	303,325,814	12,420,428	**3,828,802,933**
12	3,064,444,935	358,927,811	89,965,587	2,766,858	296,345,912	12,502,338	**3,824,953,440**

Scenarios that do not implement the extra SM (ID: 1, 2, 3, 4, 9, 11) present the same amount of bandwidth expenses, both for each OS and overall (3,828,802,933). By not having the extra SM, the clients will not experience any difference in terms of possessed stock options and so, at the start of each day, they will always buy the same amount of bandwidth, regardless of the options' life duration. The duration until expiration does not affect the outcome because, as it is observed in the simulation, should an Origin Server need to exercise its option, this need will have happened during the first 6 days that it possesses that option and in any case not after that.

On the other hand, scenarios that do implement the extra SM, have different results depending on the SO' life duration, that but most of them are higher than those of the scenarios that do not include the extra SM. In these cases, possessing SO of longer duration plays an important factor in the overall bandwidth expenses. Despite the observation that options are exercised within the first 6 days of their possession by an OS or not at all (by this OS), the OS can sell unused options even after these first 6 days and the options' buyer will most certainly exercise them immediately, thus reducing its bandwidth expenses. But, this also means that the seller now has a lower amount of SO, which may very well be necessary for exercise, in case the exchange has occurred before the observed period of the first 6 days of possession. In such a case, the seller will experience an unpredicted higher traffic in the following days and – without possessing those SO anymore- will need to acquire resources either in the secondary market by paying other OS or directly by the CDN by paying the penalized price. This is ultimately the reason why these scenarios present higher amount of bandwidth expenses of the Origin Servers despite having a better wallet balance (see metric: Wallet Change).

Scenario #12 presents the lowest overall amount of bandwidth expenses (3,824,953,440) by having a smaller percentage of acquired SO and a SO life duration

of 14 days. In this case, half of the OS experience a great decrease in bandwidth expenses because they have bought SO optimally.

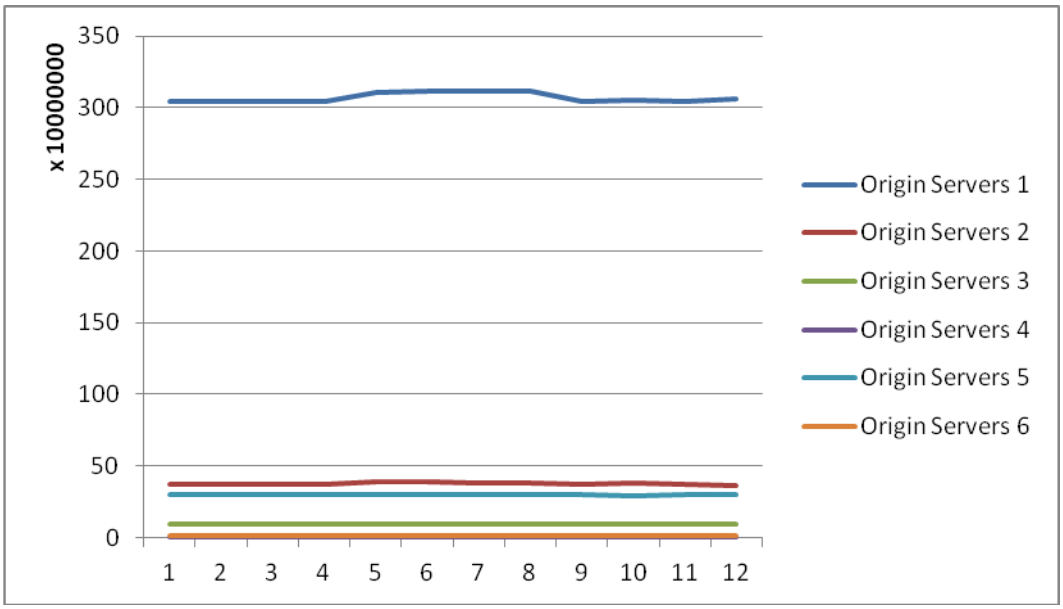

Figure 15: Expenses in Bandwidth for each OS per scenario

As the above graphs highlights, Origin Server #1 presents significantly higher expenses that all other clients, simply because of its higher traffic. This is observed across all scenarios since the element of the servers' actual traffic amount is not determined by the input parameters of our simulation but is simply predefined as a procedural element in terms of the examining the outcome of the proposed framework.

Expenses in SO per scenario for each Origin Server

Origin Servers spend a small amount of money in buying stock options compared to that of buying bandwidth resources. However, the benefit from exercising the option and getting the underlying asset when necessary and in a competitive price is reflected by examining the table below.

Table 15: Expenses in SO per scenario for each OS

	Origin Servers						
Scenario ID	1	2	3	4	5	6	Total
1	5,707,812	970,598	192,876	3,830	1,063,801	25,869	**7,964,787**
2	5,037,201	867,661	158,510	3,075	1,004,692	22,742	**7,093,881**
3	5,030,364	854,563	148,783	2,756	1,007,623	21,632	**7,065,720**
4	5,008,099	862,203	150,711	2,716	1,013,982	21,958	**7,059,669**
5	6,791,812	1,224,575	278,530	6,171	1,256,263	34,490	**9,591,840**
6	6,369,979	1,209,025	269,181	5,882	1,250,643	34,090	**9,138,799**
7	6,362,068	1,194,102	269,172	5,882	1,279,489	34,115	**9,144,828**
8	6,482,429	1,207,750	272,615	5,970	1,267,187	34,360	**9,270,312**
9	6,539,595	1,503,487	298,370	5,812	1,595,092	40,076	**9,982,431**
10	7,578,672	1,801,379	423,029	8,242	1,746,430	50,172	**11,607,924**
11	3,649,148	462,124	87,482	1,747	520,710	12,291	**4,733,502**

12	4,627,239	648,720	146,538	3,303	638,145	18,857	**6,082,802**

Each OS acquires in the beginning of each day a percentage of the predicted bandwidth traffic in the form of stock options. In our simulation this percentage varies from 5% to 15%. In scenarios without the extra SM, there is no other way of acquiring stock options. So, such scenarios' SO expenses results data differ only because of the difference in the SO price due to the difference on the SO duration until expiration. Longer SO duration results in higher SO market price but it also means that an OS has more days to keep remaining valid options that have not yet been exercised. As a result the OS will not have to buy the whole percentage of new SO in the beginning of that day.

Scenarios that involve the SM of Stock Options present higher expenses. In these scenarios, OS have a second case of spending in SO, as the exchange in the extra SM takes place after having participated in the SM of resources and having exercised any possessed options. These scenarios present a significant increase of 40.1% on average in SO expenses. It is however presented next, that the percentage increase in not transferred and reflected in the overall wallet balance of the market's participants.

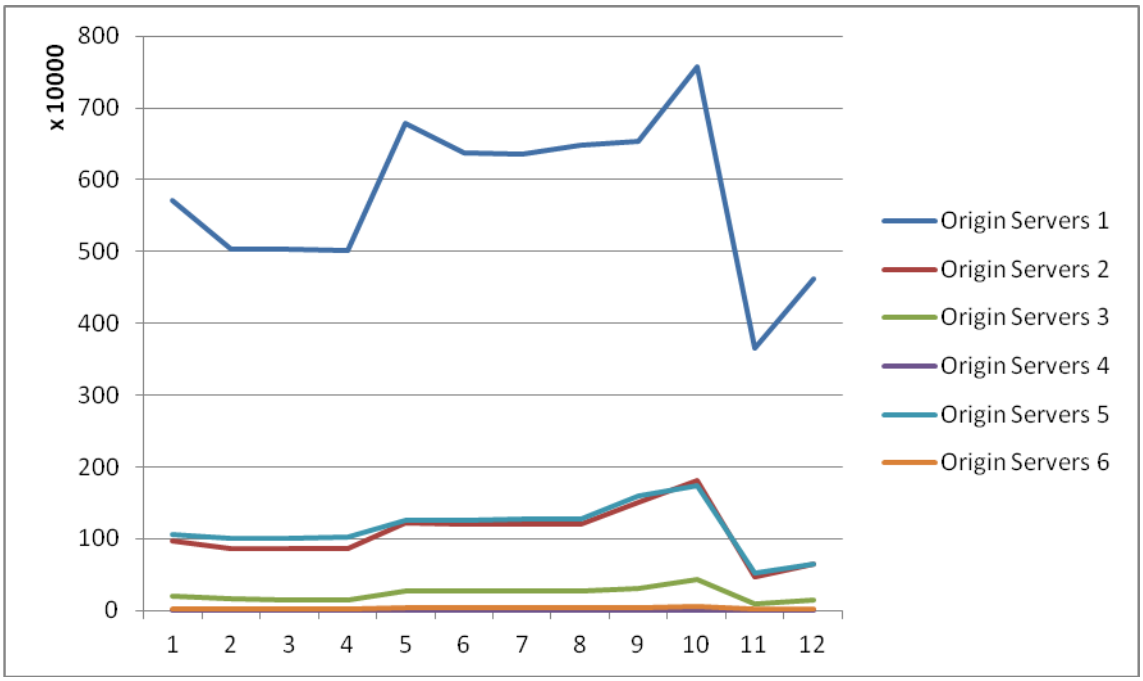

Figure 16: Expenses in SO for each OS per scenario

Wallet Change per scenario for each Origin Server

The balance of the Origin Servers' wallet is established among the most important metrics of the simulation. One of the main goals set in the proposed framework is minimizing the amount of times OS will have to buy by paying a penalized price. This strategic choice does however have an impact on the financials of the OS. The next table presents the wallet change for each Origin Server in each simulated scenario.

Table 16: Wallet Change per scenario for each OS

	Origin Servers						
Scenario ID	**1**	**2**	**3**	**4**	**5**	**6**	**Total**
1	17,221,425	-3,022,272	-2,630,053	-71,077	-10,804,948	-693,074	**-0.04**
2	17,221,425	-3,022,272	-2,630,053	-71,077	-10,804,948	-693,074	**-0.04**
3	17,221,425	-3,022,272	-2,630,053	-71,077	-10,804,948	-693,074	**-0.04**
4	17,221,425	-3,022,272	-2,630,053	-71,077	-10,804,948	-693,074	**-0.04**
5	25,057,482	-4,750,996	-3,750,377	-75,011	-15,706,053	-775,045	**-0.02**
6	21,514,785	-4,327,327	-3,735,444	-74,125	-12,602,919	-774,971	**-0.03**
7	21,193,413	-2,492,520	-4,952,297	-77,748	-12,913,009	-757,839	**-0.02**
8	21,208,997	-2,055,025	-4,951,578	-77,756	-13,366,642	-757,997	**-0.03**
9	17,221,425	-3,022,272	-2,630,053	-71,077	-10,804,948	-693,074	**-0.04**
10	23,113,052	-4,375,443	-5,027,356	-77,678	-12,865,528	-767,047	**-0.02**
11	17,221,425	-3,022,272	-2,630,053	-71,077	-10,804,948	-693,074	**-0.04**
12	4,386,477	11,188,873	-1,959,127	-82,815	-12,747,632	-785,775	**-0.02**

The most significant outcome, derived from the table, is that the implementation of the Secondary Market of Stock Options results in a slightly better overall wallet balance, but this is credited to the improvement of the financials of OS #1 in these scenarios. All other OS experience higher expenses due to the fact that, although the extra SM aids them to fulfill their needs, they have to pay a slightly higher amount in SO expenses and the options' subsequent strike price.

In terms of social welfare, scenario #12 provides the best results since 2 out of 6 OS have a positive wallet balance and the other 4 OS experience only a slight negative balance. Of course, these results are directly affected from the initial actual resources the OS need every day, where OS #1 experiences higher amounts of traffic. A more balanced distribution of the traffic among the servers would result in more evenly distributed wallet balances but, nonetheless, the overall metric is slightly improved.

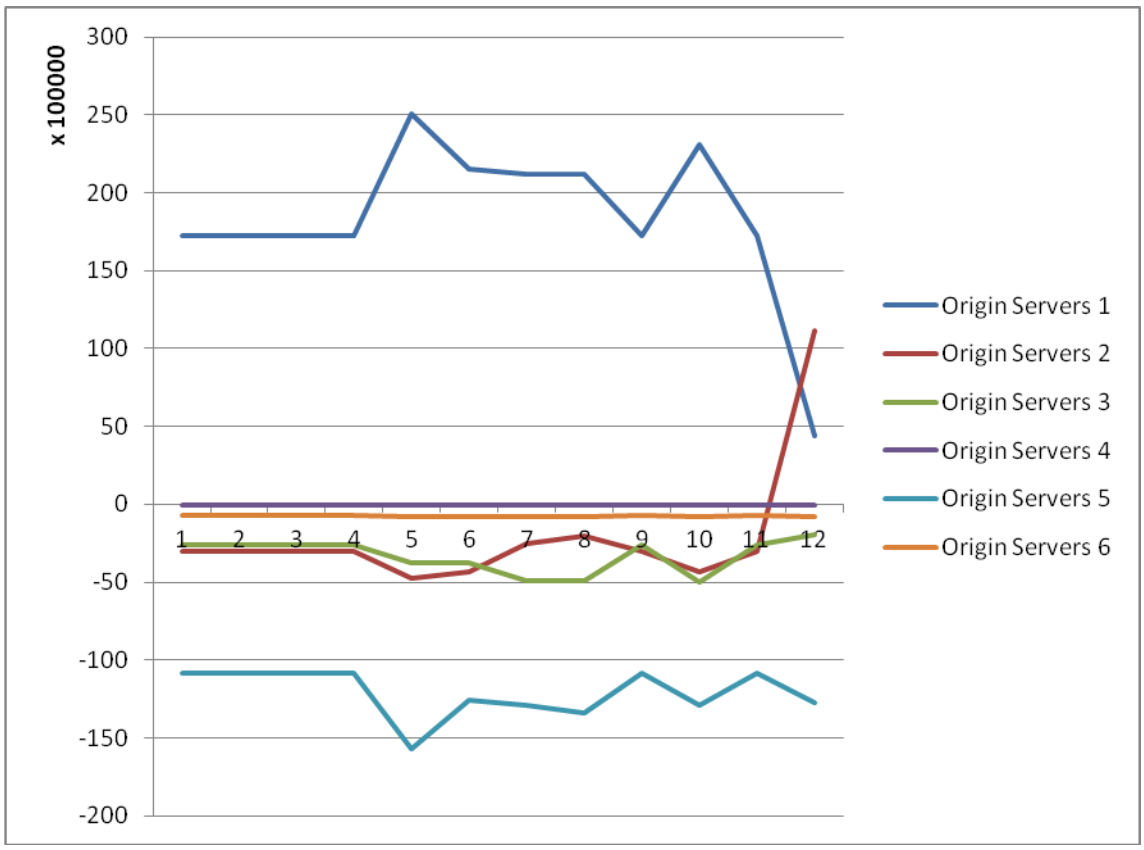

Figure 17: Wallet Change for each OS per scenario

CDN Wallet per scenario

The wallet of the CDN experiences a drastic change from the introduction of the Secondary Market of Stock Options because the CDN no longer sells frequently resources to the Origin Servers with a penalized price.

The following table presents the CDN's wallet in each scenario, with the columns depicting: the CDN's total wallet balance, the income coming from selling resources without and with penalty and also the income gained from selling stock options compared to that of selling bandwidth.

Table 17: CDN Wallet per scenario

	CDN				
Scenario ID	Total	Income from BW w/o penalty	Income from BW with penalty	Income from SO	Total Income from BW
1	4,179,148,763	3,836,767,721	342,381,042	7,964,787	4,171,183,975
2	4,175,343,312	3,835,896,814	339,446,498	7,093,881	4,168,249,431
3	4,174,356,605	3,835,868,653	338,487,952	7,065,720	4,167,290,885
4	4,174,263,941	3,835,862,602	338,401,338	7,059,669	4,167,204,272
5	4,105,453,602	3,820,543,260	284,910,343	9,479,313	4,095,974,290
6	4,168,722,781	3,881,345,818	287,376,963	9,095,394	4,159,627,387
7	4,208,687,469	3,916,727,346	291,960,123	9,144,828	4,199,542,641
8	4,209,952,362	3,922,999,245	286,953,117	9,270,312	4,200,682,050
9	4,140,301,492	3,837,842,270	302,459,223	9,980,648	4,130,320,845

10	4,077,365,167	3,838,725,456	238,639,711	11,597,786	4,065,767,382
11	4,228,722,363	3,833,536,436	395,185,927	4,733,502	4,223,988,861
12	4,174,175,466	3,831,036,242	343,139,224	6,082,802	4,168,092,665

As far as the CDN is concerned, the existence of the SM of Stock Options induces lower income for the network. Since its client can exchange options and acquire bandwidth by exercising them, they no longer turn that frequently to the CDN in order to buy extra resources in a later time. This is observed in all scenarios with the extra SM, where the “Income from BW with penalty” is significantly lower, decreased by 15.73% on average.

In several cases, the income from BW without penalty is also lower, due to the fact that valid options acquired from the SM can be kept and exercised in a later date until their expiration. On the other hand, it is observed that the ratio of the income coming from SO or BW is improved in favor of SO, since with the extra SM the CDN sells more options to the clients than previously. Furthermore, in the scenarios #9 and #10 where the initial SO buying percentage is 15%, the income deriving from options reached its highest values among all simulated scenarios. Finally, having options with longer life duration results in higher market price and in higher income from SO for the CDN. Such a comparison is presented between scenarios (#1-4) and scenarios (#5-8).

Summarizing, the existence of the Secondary Market of Stock Options, does not worsen the overall wallet of the CDN. It lowers the income derived from penalized prices but keeps the overall income at the same amount, offering a better balance while also establishing a healthier environment for the participants who no longer need to worry about paying the penalized price or even not finding available resources at all.

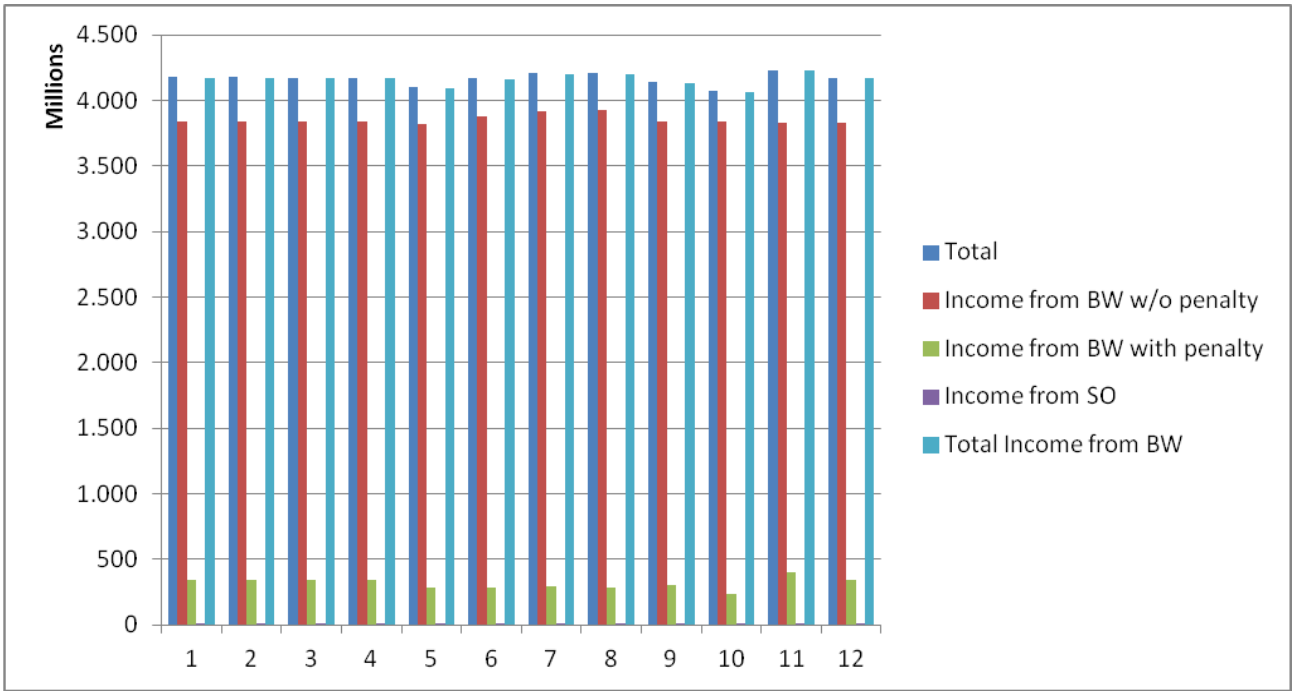

Figure 18: CDN Wallet per scenario each OS

9. CONCLUSION – FUTURE WORK

In this thesis we present an extension of the framework presented in the past work about the Capital Market Metaphor in the area of Content Distribution Networks. We are based on the framework that deals with the efficient prediction of resources by the Origin Servers, and which uses a secondary market for the exchange of resources as well as the exercise of stock options in order to minimize the prediction misses that could not be remedied in the secondary market [1] [2] [7]. The enhancement provided by our proposed framework is the introduction of an extra secondary market; a secondary market for the exchange of stock options among the Origin Servers. Having developed a simulation program we implement multiple year-long simulations of our proposed framework, observing the conducted business between the market's participants. Based on the presented and analyzed results, it has been shown that the existence of the extra secondary market has noticeably minimized the amount of times that the Origin Servers need to refer to the CDN to buy extra resources by paying the penalized price. In addition, the extra mechanism has enabled the Origin Servers to minimize their expenses by covering the cost of bought stock options that in the end did not need to exercise, by having the opportunity to sell them through the secondary market of stock options to those clients in need. The overall amount of unused stock options and their underlying total cost have been greatly decreased, since unused options are absorbed by other clients of the extra secondary market.

As far as future work on the framework's scope is regarded, this could be expanded but not limited to, to the following enhancements.

In the context of adjusting the framework's parameters, an experimentation could be the usage of other functions for calculating the price of stock options besides the Black-Scholes formula and performing comparisons between them in terms of which is more beneficial to the CDN's and OS' wallets.

The cost and pricing strategies are an important element of the market's lifecycle. In this direction, it would be interesting to observe how the market will react if the CDN has no extra resources to sell after those bought at the start of each simulated day. In this case, the secondary markets are the only way to acquire extra resources and/or stock options. The lack of the CDN's penalized price, serving until now as the maximum price of acquiring resources, will affect the selling prices in the market, based on the high demand and the competitiveness among the buyers and among the sellers.

Another experiment could be the usage of low price or high price in which the resources and stock options are to be sold from CDN based on the usage value these resources have for the Origin Servers; e.g. the CDN, based on statistics that it holds for the Origin Servers, could decide to sell either with high price or low price; that way CDN could manage the loss of income that comes from the conducted business in the extra secondary market of resources.

Furthermore, another simulation that could be performed is to check and determine whether the prediction of resources is beneficial to be performed not in a daily basis but in terms of hours e.g. every 4 or 6 hours within the day. It could also be interesting to observe the results using different pricing schemes of the CDN.

Finally, a significant experiment would be to include multiple Content Delivery Networks in the market in order to simulate an even more competitive market, which is characterized by the antagonism between the resource providers in an effort to offer competitive prices to the Origin Servers and increase their market share while also aiming to maximize their profit.

TABLE OF TERMINOLOGY

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Content Delivery Network	Δίκτυο Διανομής Περιεχομένου
Secondary Market	Δευτερογενής Αγορά
Stock Option	Δικαίωμα Προαίρεσης
Underlying Product	Υποκείμενο Προϊόν
Premium	Ασφάλιστρο (ή Τιμή Δικαιώματος)
Strike Price	Τιμή Εξάσκησης
Call Option	Δικαίωμα Αγοράς
Put Option	Δικαίωμα Πώλησης
Risk-free Rate	Επιτόκιο δίχως κίνδυνο

ABBREVIATIONS - ACRONYMS

UoA	University of Athens
NKUA	National and Kapodistrian University of Athens
CDN	Content Delivery Network
OS	Origin Server
SO	Stock Option
SOM	Stock Options Mechanism
SM	Secondary Market
SMM	Secondary Market Mechanism
PRS	Predictive Reservation Scheme
KRE	Kernel Regression Estimator
DDoS	Distributed Denial of Service
IRDM	Inertia Region Detection Mechanism
DEDM	Deviation Early Detection Mechanism
IRRMM	Initial Resource Reservation Monitoring Mechanism
QoS	Quality of Service
BW	Bandwidth
GB	Gigabyte
RAM	Random Access Memory

ANNEX I: PRICING POLICIES AMONG CDN PROVIDERS

Major Content Delivery Network companies compete in the following categories:

- Range of services (e.g. multimedia services, live / dynamic content delivery, content security etc.)
- Market share (e.g., coverage areas, availability of service etc.)
- Cost and Pricing of provided services
- Support Services (e.g. 24/7 support, backup, compensation policy etc.)

Most CDN providers offer the following pricing policy plans for their services:

- Charge of a fixed amount depending on the volume of resource use. Types of tiers are typically used, where service usage is low, medium or high, and customers are charged at a fixed price based on these levels.
- Appropriate analogy of the charge based on exactly the usage range of the service by the customers, who pay exactly for exactly what they have used. This type of pricing is called “Pay As You GO” (PAYG).
- Free service provision: The service is provided at no cost but is nonetheless governed by a set of limitations (usage period, data volume, number of resources).
- Pipeline pricing (also called Burstable Billing): It is a pricing plan based on the 90-95% rate of agreed time to serve possible variations in the customer's maximum needed volume. It is an ideal charging plan for customers with a constant volume of required resources on a regular basis and whose needs change only at specific time period, the total of which do not exceed 5-10% of the total usage period defined by the service contract. Should the period of high demand exceed the 10% of the service usage period, then the customer is subject to a significant additional charge in order to obtain the requested resources. An example that highlights the usefulness of this policy is a website that offers live streaming of sporting events. Since the majority of sports games are scheduled on the weekend, the traffic of such a website is usually low from Monday until Friday and therefore can benefit if it chooses a pipeline pricing model based on its traffic on 2 out of 7 days of the week instead of choosing a flat charge as if it experienced the same, weekend traffic volume on all 7 days of the week.
- Mix of the aforementioned pricing policies.

<u>The Case Study of Google Storage</u>

Google is ranked among the top storage providers, both at the level of simple storage provision with the ‘Google Drive’ service and at the more high-end level with the ‘Google Cloud’ service, which offers additional functionality to its customers.

In summary, Google offers four tiers of storage provision services, mostly differentiating on the distance between the storage space and the customer, called Storage Classes [10]:

- Multi-Regional Storage
- Regional Storage
- Nearest Storage
- Coldline Storage

All Storage Classes offer the same throughput, low latency and high durability of the data.
The difference on their cost depends on their different properties, which are displayed in the following table.

Table 18: Storage Classes Properties

Storage Class	Properties	Use Cases
Multi-Regional Storage	Availability > 99.99% SLA: 99.95% Cost of GB per month: (0,026$) Geo-Redundancy The data is spread across multiple locations all over the world.	This class is used for storing the most important and frequently accessed data (hot objects) with direct access from all over the world, regardless of the location and the number of requests. Examples of use include websites' content, online applications and games, multimedia such as streaming videos etc.
Regional Storage	Availability: 99.99% SLA: 99.9% Cost of GB per month: (0,02$) The data is kept in Storage spaces in related geographical areas and in close proximity to the client. Redundancy: only between the areas of the Region.	Storing data of frequent access but with no request for high availability in terms of geo-redundancy. They are kept in a space related to the Cloud for cases like data analysis, data mining etc. (Data Analytics).
Nearline Storage	Availability: 99,9% SLA: 99,9% Cost of GB per month: (0,01$) Extra charge for data recovery as well as higher operating costs. Minimum data preservation period: 30 days	Used for storing data accessed less than once a month. Useful for requests such as those that arise for Backup purposes and procedures.
Coldline Storage	Availability: 99,9% SLA: 99,9% Cost of GB per month: (0,007$) Extra charge for data recovery as well as higher operating costs that those of the Nearline Storage Class.	Useful for data with rare access need, perhaps less than once a year. Typically, it is selected for data to be archived or for data needed for Disaster Recovery Readiness Plan.

	Minimum data preservation period: 90 days	

The pricing policy and billing of Google Cloud Storage is based on the following features [11]:

- Data Storage: The charge is based on the size of the customer's data, which will be stored in company's resources depending on the desired physical location of the storage. Based on the desired location, the customer chooses a Storage class to store his data.

- Network Usage: Charge is applied when data traffic rate in the client's storage occurs, for example in cases of copy or move of the data to other locations or for use by third-party services. The network usage is divided into the following categories:
    - Network Use between Buckets: Refers to moving data between different buckets of the customer, using the network for in-house classifications of the data stored on the infrastructure. The amount of billing per GB varies, depending on the volume of data and the Regions of the source and the destination buckets. If those are in the same location, there is no charge. But if they are in a different area of the same Multi-Region, for example on the two United States of America coasts, there is a flat charge of 0.01 $ per GB, regardless of the volume of data. Finally, the charging fee increases in cases of network use between buckets of different worldwide geographical regions, for example between Europe (eu) and Asia (asia), Western Europe (eu-west1) and South-eastern Australia (Australia-southeast1). The charge in these cases starts at 0.12 $ per GB for a data volume between 0-1 TB, ranges at 0.11 $ per GB for a volume between 1-10 TB and is set at 0.08 $ per GB for more than 10 TB of data.
    - Specialty Network Services: Cloud CDN, CDN Interconnect, Cloud Interconnect, Direct Peering etc.
    - Free Network Usage: Free data traffic is provided in cases where Google Cloud Platform buckets are located topologically in the same area as the customer's instance Engine. An example of free network use is when accessing data that is stored in the "eu" bucket while the App Engine of the GCP service is located in the "eu-west1" region and therefore within the wider European coverage area.
    - Generic Network Usage: Refers to the generic use case of network date, namely to Read/Write operations between Google Cloud Storage buckets, for example when moving data between storage buckets and the local disk drive of the client's computer. General network use includes cases, which are not covered by the other categories.

- Operational Usage: Refers to performing operations on data held in any of the four Storage Classes. Depending on the Storage Class and the type of work that the client wishes to perform, the cost also arises, with the charging base being based on 10000 data operations. Operations are categorized into 3 tiers: Class A Operations, Class B Operations and Free Operations, according to the following table. Class A operations are more common and are charged between 0.05 $ / GB and 0.10 $ / GB depending on the Storage Class. Regional and Multi-

Regional classes are always more expensive than the Nearline storage class. Finally, Coldline Storage class is subject to the lowest possible charge.

Table 19: Classification of Google Cloud Storage Operations

Class A Operations	Class B Operations	Free Operations
storage.*.insert storage.*.patch storage.*.update storage.*.setIamPolicy storage.buckets.list storage.notifications.delete storage.objects.compose storage.objects.copy storage.objects.list storage.objects.rewrite storage.objects.watchAll storage.*AccessControls.delete	storage.*.get storage.*.getIamPolicy storage.*.testIamPermissions storage.*AccessControls.list storage.notifications.list Each object notification	storage.channels.stop storage.buckets.delete storage.objects.delete
GET Service GET Bucket (when listing objects in a bucket) PUT POST	GET Bucket (when retrieving bucket configuration) GET Object HEAD	DELETE
SetStorageClass		Delete

- Data Retrieval and Early Deletion services fees): The charge for data retrieval and early deletion services applies only to Nearline and Coldline Storage classes, namely when a Storage Class with features such as long-term data retention and at the same time infrequent access has been selected by the client. The cost of data recovery is applied in addition to network usage costs and is set at 0.01 $ / GB and 0.05 $ / GB for Nearline και Coldline Storage respectively. If data is deleted from the storage prior to the minimum retention period, which is set to 30 days for the Nearline and 90 days for the Coldline Storage respectively, then the charge is calculated as if the data was deleted just after that period had expired and not earlier than its expiration date.

ANNEX II: SOURCE CODE OF THE SIMULATION PROGRAM

The program used for the simulation was created in Java.

The following methods are presented.

- ***fillEmptyData***
- ***performAuction (for SO)***
- ***makeSale (for SO)***
- ***exerciseStockOptions***
- ***performAuction (for Resources)***
- ***makeSale (for Resources)***
- ***concludeData***

Analytically:

- ***fillEmptyData***

This method is used at the start of the simulation scenario to calculate the needs of the Origin Servers in terms of both resources and also stock options of resources to be bought from CDN in the beginning of the day.

```
public static ArrayList<OriginServer> fillEmptyData(int
day,ArrayList<OriginServer> currentDayOs,double soPercent,int
soDuration,boolean soExchange){


          ContentDeliveryNetwork cdn = null;


          if(day == 1) // if it is the 1st day of the simulation then
previousDayOs is null/empty
          {


               for(int i=0; i < currentDayOs.size(); i++){


                    //at first day , initially ordered BW  = predicted
traffic


     currentDayOs.get(i).setInitialTraffic(currentDayOs.get(i).getPredictedTra
```

```
ffic());


                    //calculate the unused traffic (BW) of the day
                    int unusedTraffic =
currentDayOs.get(i).getInitialTraffic() -
currentDayOs.get(i).getRealTraffic();



     currentDayOs.get(i).setInitialUnusedTraffic(unusedTraffic);


                    //we calculate the normal/high and low price of the CDN
                    double normalPrice =
FunctionsCDN.calculateCDNPrice(currentDayOs.get(i).getInitialTraffic());


                    double highPrice = normalPrice*1.2;


                    double lowPrice = normalPrice*0.95;


                    //stock options to be purchased on first day
                    int stockOptions = (int)
Math.round((currentDayOs.get(i).getPredictedTraffic()*soPercent));


                    currentDayOs.get(i).setHighPrice(highPrice);
                    currentDayOs.get(i).setNormalPrice(normalPrice);
                    currentDayOs.get(i).setLowPrice(lowPrice);
                    currentDayOs.get(i).setPredictedSO(stockOptions);


                    CdnWallet cdnWallet = new CdnWallet();
                    cdnWallet.setDay(day);
                    cdnWallet.setOsId(currentDayOs.get(i).getId());
                    cdnWallet.setPrice(normalPrice);
                    cdnWallet.setPriceFlag("N");
```

```
     cdnWallet.setResources(currentDayOs.get(i).getInitialTraffic());
                    cdnWallet.setResourceType("BW");
                    cdnWallet.setSource("OS");

     cdnWallet.setWallet(Precision.round(normalPrice*currentDayOs.get(i).getIn
itialTraffic(),5));

                    //we update the CDN Wallet after the purchase of the BW
                    DBControl.insertCDNWallet(cdnWallet);



                    //we retrieve the cost plan based on the volume of stock
option to be purchased
                    String costPlan = FunctionsCDN.getCostPlan(stockOptions);

                    cdn = DBControl.retrieveCDNDetails(costPlan, soDuration);

                    StockOption so = new StockOption();

                    so.setDayPurchased(day);
                    so.setDaysExpire(soDuration);
                    so.setOsId(currentDayOs.get(i).getId());
                    so.setSoNumber(stockOptions);
                    so.setSoBought(stockOptions);
                    so.setSoCost(cdn.getStockOptionPrice()*stockOptions);
                    so.setValid(true);
                    so.setPurchaseOrigin("CDN");

                    cdnWallet = new CdnWallet();
                    cdnWallet.setDay(day);
```

```
                    cdnWallet.setOsId(currentDayOs.get(i).getId());
                    cdnWallet.setPrice(cdn.getStockOptionPrice());
                    cdnWallet.setPriceFlag("-");
                    cdnWallet.setResources(stockOptions);
                    cdnWallet.setResourceType("SO");
                    cdnWallet.setSource("OS");

    cdnWallet.setWallet(Precision.round(cdn.getStockOptionPrice()*stockOption
s,5));

                    //we update the CDN Wallet after the purchase of the SOs
                    DBControl.insertCDNWallet(cdnWallet);

                    logger.info(currentDayOs.get(i).toString());

                    logger.info(so.toString());
                    // update os to DB
                    DBControl.updateOriginServer(currentDayOs.get(i),day);
                    // to add so to DB
                    DBControl.insertSOData(so);
               }
          }
          else {

               OriginServer previousDayOS  = null;

               for(int i = 0; i< currentDayOs.size(); i ++){

                    int osId = currentDayOs.get(i).getId();

                    previousDayOS =
```

```
DBControl.retrieveOriginServerDetails(osId, day-1);


                    //in case any traffic remained from previous day
                    if(previousDayOS.getTrafficTransfer()>0){


                         //we order in BW the difference if previous day we
had BW to be transferred today
                         if((currentDayOs.get(i).getPredictedTraffic()-
previousDayOS.getTrafficTransfer())>0){

     currentDayOs.get(i).setInitialTraffic(currentDayOs.get(i).getPredictedTra
ffic()-previousDayOS.getTrafficTransfer());
                         }
                         else{ //we do not order any more traffic since the
OS is covered from the previous
                              currentDayOs.get(i).setInitialTraffic(0);
                         }


                    }else{ //if no traffic remained from previous day we
order traffic equal with the prediction

     currentDayOs.get(i).setInitialTraffic(currentDayOs.get(i).getPredictedTra
ffic());
                    }



                    //we calculate the initially unused BW
                    //if the previous day the OS didnt have any traffic
remaining
                    if(previousDayOS.getTrafficTransfer()<=0){
                         //The unused traffic is the result of what the OS
initially ordered minus the real traffic of the day
```

```
     currentDayOs.get(i).setInitialUnusedTraffic(currentDayOs.get(i).getInitia
lTraffic()-currentDayOs.get(i).getRealTraffic());
                    }else{
                         //the unused traffic is the result of what remained
the previous day plus the initial traffic ordered minus the real traffic of
the day

     currentDayOs.get(i).setInitialUnusedTraffic(previousDayOS.getTrafficTrans
fer()+currentDayOs.get(i).getInitialTraffic()-
currentDayOs.get(i).getRealTraffic());
                    }

                    //we set the wallet change for the current day to 0
                    currentDayOs.get(i).setWalletChange(0);

                    //we calculate the normal/high and low price of the CDN
                    double normalPrice =
FunctionsCDN.calculateCDNPrice(currentDayOs.get(i).getInitialTraffic());
                    double highPrice = normalPrice*1.2;
                    double lowPrice = normalPrice*0.95;

                    currentDayOs.get(i).setHighPrice(highPrice);
                    currentDayOs.get(i).setNormalPrice(normalPrice);
                    currentDayOs.get(i).setLowPrice(lowPrice);



                    CdnWallet cdnWallet = new CdnWallet();

                    if(currentDayOs.get(i).getInitialTraffic()>0)
                    {
                         cdnWallet.setDay(day);
                         cdnWallet.setOsId(currentDayOs.get(i).getId());
```

```
                        cdnWallet.setPrice(normalPrice);
                        cdnWallet.setPriceFlag("N");

     cdnWallet.setResources(currentDayOs.get(i).getInitialTraffic());
                        cdnWallet.setResourceType("BW");
                        cdnWallet.setSource("OS");

     cdnWallet.setWallet(Precision.round(normalPrice*currentDayOs.get(i).getInitialTraffic(),5));

                        //we update the CDN Wallet after the purchase of the BW
                        DBControl.insertCDNWallet(cdnWallet);
                    }

                    //we estimate the SO to be purchased
                    //if we have SOs transferred from the previous day
                    if(previousDayOS.getStockOptionsTransferred()>0){

                        //we calculate the remaining SOs of the previous day compared to the current's day bought traffic (BW-Resources)
                        double soTransferedPercent = ((double)previousDayOS.getStockOptionsTransferred()/currentDayOs.get(i).getInitialTraffic());

                        //we adjust the percentage of SOs to be bought from CDN
                        double tempSoPercent = stockOptionPercentAdjustment(osId,day,soPercent);

                        //if the percent of the remaining SOs is less than the soPercent of the traffic bought
                        if(soTransferedPercent<=tempSoPercent){
```

```
                                //OS will buy SOs in order to cover the
percentage needed to cover at least his needs if needed
                                double soPercentBuy = tempSoPercent-
soTransferedPercent;


                                int stockOptions = (int)
Math.round((currentDayOs.get(i).getInitialTraffic()*soPercentBuy));



     currentDayOs.get(i).setPredictedSO(stockOptions);


                                //we retrieve the cost plan based on the volume
of stock option to be purchased
                                String costPlan =
FunctionsCDN.getCostPlan(stockOptions);


                                cdn = DBControl.retrieveCDNDetails(costPlan,
soDuration);


                                StockOption so = new StockOption();


                                so.setDayPurchased(day);
                                so.setDaysExpire(soDuration);
                                so.setOsId(currentDayOs.get(i).getId());
                                so.setSoNumber(stockOptions);
                                so.setSoBought(stockOptions);


     so.setSoCost(cdn.getStockOptionPrice()*stockOptions);
                                so.setValid(true);
                                so.setPurchaseOrigin("CDN");
```

```
                                cdnWallet = new CdnWallet();
                                cdnWallet.setDay(day);
                                cdnWallet.setOsId(currentDayOs.get(i).getId());
                                cdnWallet.setPrice(cdn.getStockOptionPrice());
                                cdnWallet.setPriceFlag("-");
                                cdnWallet.setResources(stockOptions);
                                cdnWallet.setResourceType("SO");
                                cdnWallet.setSource("OS");

     cdnWallet.setWallet(Precision.round(cdn.getStockOptionPrice()*stockOption
s,5));


                                //we update the CDN Wallet after the purchase
of the SOs
                                DBControl.insertCDNWallet(cdnWallet);


                                logger.info(currentDayOs.get(i).toString());


                                logger.info(so.toString());
                                // update os to DB

     DBControl.updateOriginServer(currentDayOs.get(i),day);
                                // to add so to DB
                                DBControl.insertSOData(so);


                           }


                      } else{ // if no SOs remained the previous day


                           //OS will buy a percentage of SOs soPercent based on
the initial bought Traffic(BW)
                           int stockOptions = (int)
```

```
Math.round((currentDayOs.get(i).getInitialTraffic()*soPercent)+0.5);


                        currentDayOs.get(i).setPredictedSO(stockOptions);


                        //we retrieve the cost plan based on the volume of
stock option to be purchased
                        String costPlan =
FunctionsCDN.getCostPlan(stockOptions);


                        cdn = DBControl.retrieveCDNDetails(costPlan,
soDuration);


                        StockOption so = new StockOption();


                        so.setDayPurchased(day);
                        so.setDaysExpire(soDuration);
                        so.setOsId(currentDayOs.get(i).getId());
                        so.setSoNumber(stockOptions);
                        so.setSoBought(stockOptions);


     so.setSoCost(cdn.getStockOptionPrice()*stockOptions);
                        so.setValid(true);
                        so.setPurchaseOrigin("CDN");



                        cdnWallet = new CdnWallet();
                        cdnWallet.setDay(day);
                        cdnWallet.setOsId(currentDayOs.get(i).getId());
                        cdnWallet.setPrice(cdn.getStockOptionPrice());
                        cdnWallet.setPriceFlag("-");
                        cdnWallet.setResources(stockOptions);
```

```
                        cdnWallet.setResourceType("SO");
                        cdnWallet.setSource("OS");

    cdnWallet.setWallet(cdn.getStockOptionPrice()*stockOptions);

                        //we update the CDN Wallet after the purchase of the
SOs
                        DBControl.insertCDNWallet(cdnWallet);

                        logger.info(currentDayOs.get(i).toString());

                        logger.info(so.toString());
                        // update os to DB

    DBControl.updateOriginServer(currentDayOs.get(i),day);
                        // to add so to DB
                        DBControl.insertSOData(so);

                    }
                }

            }

            return currentDayOs;
    }
```

- **performAuction (for SO)**

This method is used to perform the Auction of Stock Options in the Secondary Market.

```
/*
 * Method used to perform the SO Auction
```

```
 */
public static ArrayList<OriginServer> performAuction(ArrayList<OriginServer>
currentDayOs,int day){


          //we initialize the variables for the buyers/sellers of SOs in SM


          ArrayList<OriginServer> buyersList = new ArrayList<OriginServer>();
          ArrayList<OriginServer> sellersList = new ArrayList<OriginServer>();


          ArrayList<OriginServer> osList = new ArrayList<OriginServer>();


          //we identify the potential buyers/sellers for the auction in the
Secondary Market
          for(int i=0; i<currentDayOs.size(); i++){


               int remainingTrafficAfterAuction = (-
1)*currentDayOs.get(i).getRemainingTrafficAfterAuction();


               int stockOptionsTransfered =
currentDayOs.get(i).getStockOptionsTransferred(); // these are the total SOs
of the OS to possible transfer to next day


               if(remainingTrafficAfterAuction>0){


                    currentDayOs.get(i).setOsAction(-1); // we set the action
of the OS as a buyer


 // we set the initial value for the Traffic AFTER the auction


                    currentDayOs.get(i).setSmSellingPrice(0); // we set the
sm selling price to 0 since the OS is not a seller


                    DBControl.updateOriginServer(currentDayOs.get(i), day);
```

```
// we update the data in the DB


                    buyersList.add(currentDayOs.get(i)); // we add the OS to
the potential buyers
                }
               else if(remainingTrafficAfterAuction<=0 &&
stockOptionsTransfered>0){


                currentDayOs.get(i).setOsAction(1);  // we set the action of
the OS as a seller
                     // we set the initial value for the Traffic AFTER the
auction


     //currentDayOs.get(i).setRemainingTrafficAfterAuction(currentDayOs.get(i)
.getInitialUnusedTraffic());
                DBControl.updateOriginServer(currentDayOs.get(i), day); // we
update the data in the DB
                    sellersList.add(currentDayOs.get(i)); // we add the OS to
the potential sellers
               }
               else if(remainingTrafficAfterAuction>=0 &&
stockOptionsTransfered==0){
                currentDayOs.get(i).setOsAction(0);  // the OS is neither a
seller nor a buyer
                DBControl.updateOriginServer(currentDayOs.get(i), day); // we
update the data in the DB
               }
          }


          Logger.info("Number of Potential
{Buyers,Sellers}:"+"{"+buyersList.size()+","+sellersList.size()+"}");


          //we check if we have at least 1 buyer and 1 seller
```

```
          if(!buyersList.isEmpty() && !sellersList.isEmpty()){


               //we get the SOs to be exchanged by ascending order of
days_till_expiration
               ArrayList<StockOption> sellersSOList =
getSellerSo(sellersList);


               //prepare the sellers data for auction
               sellersList = updateSellersData(sellersSOList,sellersList,day);


               for(int j=0; j < sellersList.size(); j++){ // we find the
potential buyers for the respective seller


                    int potentialBuyers = 0 ;


                    for(int i=0; i<buyersList.size(); i++){
                         if(evaluation(buyersList.get(i),sellersList.get(j)))
                              potentialBuyers++;
                    }


                    if(potentialBuyers>0){
                         Logger.info("The buyers for Seller
{"+sellersList.get(j).getId()+"} are "+potentialBuyers);


                         //we proceed with the buy splitted fairly between
all buyers


                         //we keep the total number of SOs that the Seller is
selling
                         int totalSellingSOs =
sellersList.get(j).getRemainingTrafficAfterAuction();
```

```
                        //we keep the total number of SOs that need to be
bought
                        int totalBuyingSOs = 0;


                        for(int k=0; k<buyersList.size(); k++){
                             totalBuyingSOs +=
buyersList.get(k).getRemainingTrafficAfterAuction();
                        }


                        totalBuyingSOs = totalBuyingSOs*(-1);


                        Logger.info("SOs to be sold "+totalSellingSOs+" and
"+totalBuyingSOs + " SOs need to be bought in SM.");


                        if(totalSellingSOs>=totalBuyingSOs &&
totalBuyingSOs>0){


                             Logger.info("The amount of SOs to be sold is
more (or the same) than the amount "
                                       + "needed to be bought. The
"+buyersList.size()+" buyers will buy all the SOs they need from seller
{"+sellersList.get(j).getId()+"}."
                                       + " Seller
{"+sellersList.get(j).getId()+"} will remain with "+ (totalSellingSOs-
totalBuyingSOs) + " traffic in SO.");


                             for(int i = 0; i < buyersList.size(); i++)
                                  makeSale(sellersList.get(j),
buyersList.get(i), day,0,sellersSOList);


                           break; // we exit the loop the rest of the
sellers(if any) will not sell their SOs
```

```
                        }else if (totalSellingSOs<totalBuyingSOs){


                                double sellingPercentage =
(double)totalSellingSOs/totalBuyingSOs;


                                sellingPercentage =
Precision.round(sellingPercentage,15);


Logger.info("The amount of SOs to be sold is less than the amount needed to be
bought. The "+buyersList.size()+" buyers "
                                        + "will buy "+
Precision.round(sellingPercentage*100,15) + "% of their needs (traffic in SOs)
from Seller {"+sellersList.get(j).getId()+"} . The Seller will not have any
SOs remaining.");


Logger.info("Selling Percentage:"+sellingPercentage);
for(int i = 0; i < buyersList.size(); i++){


                                int buyersNeeds =
buyersList.get(i).getRemainingTrafficAfterAuction()*(-1);


                                int partialBought = (int)
Math.round(sellingPercentage*buyersNeeds);
                                if(partialBought==0){
                                     partialBought++;
                                }
if(totalSellingSOs-partialBought>=0){


Logger.info("Buyer {"+buyersList.get(i).getId()+"} will buy:"+partialBought);
                                        makeSale(sellersList.get(j),
buyersList.get(i), day,partialBought,sellersSOList);
                                        //we set the remaining SOs after
auction for the seller to 0
```

```
     sellersList.get(j).setRemainingTrafficAfterAuction(0);
                                    }else
                                         Logger.info("Buyer
{"+buyersList.get(i).getId()+"} not will buy :"+partialBought+" from Seller
due to insufficient SOs");

     totalSellingSOs -= partialBought;
     DBControl.updateOriginServer(sellersList.get(j), day);

                               }
                          }
                     }
                }
           }else
             Logger.info("Not enough Buyers or/and Sellers in the Secondary
Market to perform the auction.");

           //here we return the list with the updated data after the auction of
the day
           if(sellersList.size()!=0){
                for(int i = 0; i<sellersList.size(); i++)
                     osList.add(sellersList.get(i));
           }

           if(buyersList.size()!=0){
                for(int i = 0; i<buyersList.size(); i++)
                     osList.add(buyersList.get(i));
           }

           for(int i=0; i<osList.size(); i++)
                Logger.info(osList.get(i).toString());
```

```
            return osList;
        }
```

- **makeSale (for SO)**

This function is responsible for the simulation of the sale of Stock Options between the Origin Servers.

```
        /*
         * This function is responsible to perform the sale
         */
        private static void makeSale(OriginServer seller,OriginServer buyer,int day,
int partial, ArrayList<StockOption> soList){


            if(partial>0){


                //we update the bucket with the SOs for the seller
                for(int i=0; i < soList.size();i++){
                    if(soList.get(i).getOsId()==seller.getId()){


                        soList.get(i).setSoNumber(soList.get(i).getSoNumber()-
partial);


                        DBControl.updateStockOptions(soList.get(i));


                        Logger.info("Seller {"+seller.getId()+"} remains with "+
soList.get(i).getSoNumber()+ " SOs");


                        break;
                    }
                }
```

```
                //update the remaining traffic after auction for buyer and seller

     seller.setRemainingTrafficAfterAuction(seller.getRemainingTrafficAfterAuctio
n() - partial);

     buyer.setRemainingTrafficAfterAuction(buyer.getRemainingTrafficAfterAuction(
) + partial);


                //we update the exchanged SO for the seller and the buyer
                seller.setExchangedSO(seller.getExchangedSO()+partial);
                buyer.setExchangedSO(buyer.getExchangedSO()+partial);


                //update the wallet for the OSs
                seller.setWalletChange(seller.getWalletChange() +
partial*seller.getSmSellingPrice());
                buyer.setWalletChange(buyer.getWalletChange()  -
partial*seller.getSmSellingPrice());


           }else{


                int buyerTrafficNeeds = buyer.getRemainingTrafficAfterAuction()*(-
1);


                //we update the bucket with the SOs for the seller
                for(int i=0; i < soList.size();i++){
                     if(soList.get(i).getOsId()==seller.getId()){


                          soList.get(i).setSoNumber(soList.get(i).getSoNumber()-
buyerTrafficNeeds);


                          DBControl.updateStockOptions(soList.get(i));


                          Logger.info("Seller {"+seller.getId()+"} remains with "+
```

```
soList.get(i).getSoNumber()+ " SOs");


                        break;

                    }

                }



                //update the remaining traffic after auction for buyer and seller

     seller.setRemainingTrafficAfterAuction(seller.getRemainingTrafficAfterAuctio
n() - buyerTrafficNeeds);

     buyer.setRemainingTrafficAfterAuction(buyer.getRemainingTrafficAfterAuction(
) + buyerTrafficNeeds);

                //we update the exchanged SO for the seller and the buyer

                seller.setExchangedSO(seller.getExchangedSO()+buyerTrafficNeeds);

                buyer.setExchangedSO(buyer.getExchangedSO()+buyerTrafficNeeds);


                //update the wallet for the OSs

                seller.setWalletChange(seller.getWalletChange() +
buyerTrafficNeeds*seller.getSmSellingPrice());

                buyer.setWalletChange(buyer.getWalletChange()  -
buyerTrafficNeeds*seller.getSmSellingPrice());


          }

          //update also the DB data

          DBControl.updateOriginServer(seller, day);

          DBControl.updateOriginServer(buyer, day);

     }
```

- **exerciseStockOptions**

This function is used to simulate the exercise of Stock Options by the Origin Servers.

```
/*
       * Method used for the OS to exercise their SOs
       */
      @SuppressWarnings("unchecked")
      public static ArrayList<OriginServer>
exerciseStockOptions(ArrayList<OriginServer> currentDayOs,int day) {



       for(int i=0; i<currentDayOs.size(); i++){
           //we check if we have unused BW


           if(currentDayOs.get(i).getRemainingTrafficAfterAuction()<0){ //if we do
not have any BW we check i we have SOs to exercise


                 Logger.info("OS #"+currentDayOs.get(i).getId()+" will exercise
Stock Options if he has any.");


                 int remainingTrafficAfterAuction = (-
1)*currentDayOs.get(i).getRemainingTrafficAfterAuction();
                 int osId = currentDayOs.get(i).getId();


                 ArrayList<StockOption> soList =
DBControl.retrieveStockOptionsByOS(osId);


                 if(!soList.isEmpty()){


                       //we assume that all the valid SOs of the OS are eligible for
transfer the next day
                       for(int k=0; k<soList.size(); k++)
```

```
     currentDayOs.get(i).setStockOptionsTransferred(currentDayOs.get(i).getStockO
ptionsTransferred()+soList.get(k).getSoNumber());



                    Logger.info("OS #"+currentDayOs.get(i).getId()+" has
"+currentDayOs.get(i).getStockOptionsTransferred()+" Total Stock Options to
exercise.");
                    //we sort the list based on the expiration day lowest to
highest (days till expire)

                    Collections.sort(soList);

                    int trafficNeeds = remainingTrafficAfterAuction;

                    Logger.info("OS #"+currentDayOs.get(i).getId()+" needs
"+trafficNeeds+" Total Resources.");
                    //we exercise SOs until traffic covered
                    for(StockOption so : soList){

                         if(trafficNeeds>0){
                         // if the SO number is not enough to cover all traffic
for the day
                              if(so.getSoNumber()<trafficNeeds){

                                trafficNeeds = trafficNeeds - so.getSoNumber();
// we check if any SOs Remain after excersize

                                //we excersize the SOs

currentDayOs.get(i).setRemainingTrafficAfterAuction(currentDayOs.get(i).getRemain
ingTrafficAfterAuction()+so.getSoNumber());

                                //we subtract the amount of the excersized SOs
```

```
from the sum to be transfered the next day

currentDayOs.get(i).setStockOptionsTransferred(currentDayOs.get(i).getStockOption
sTransferred()-so.getSoNumber());

                                logger.info("OS #"+currentDayOs.get(i).getId()+"
got "+so.getSoNumber()+" Resources by exercising Stock Options.");
                                logger.info("OS #"+currentDayOs.get(i).getId()+"
needs "+trafficNeeds+" Total Resources After Stock Options exercise.");

                                so.setSoNumber(0); // we exercised all the SO

                                DBControl.updateStockOptions(so);

                            }
                              // if the SO number is enough to cover all traffic
for the day
                            else if (so.getSoNumber()>=trafficNeeds){

                                 //we subtract the amount of the excersized SOs
from the available sum for the next day

currentDayOs.get(i).setStockOptionsTransferred(currentDayOs.get(i).getStockOption
sTransferred()-trafficNeeds);

                                 logger.info("OS
#"+currentDayOs.get(i).getId()+" got "+trafficNeeds+" Resources by exercising
Stock Options.");

                                trafficNeeds = trafficNeeds - so.getSoNumber();
// we check if any resources of this SO remain after exercise

                                so.setSoNumber(trafficNeeds*(-1)); // the so
```

```
amount is reduced by the traffic needed
                                    //we exercise the SOs and the OS doesn't need
any resources

currentDayOs.get(i).setRemainingTrafficAfterAuction(0);

                                    DBControl.updateStockOptions(so);

                                    logger.info("OS #"+currentDayOs.get(i).getId()+"
needs 0 Total Resources After Stock Options exercise.");

                               }
                              }else break;

                         }
                         //currentDayOs.get(i).setInitialUnusedTraffic(trafficNeeds);
                         logger.info("OS #"+currentDayOs.get(i).getId()+" needs to buy
"+ currentDayOs.get(i).getRemainingTrafficAfterAuction()*(-1)+" Total Resources
(in Stock Options) via SM.");
                    }

                    DBControl.updateOriginServer(currentDayOs.get(i), day);

                }else{ // all the stock options of the OS can potentially be
exchanged in SM
                    logger.info("OS #"+currentDayOs.get(i).getId()+" will not be
exercising Stock Options....");
                    ArrayList<StockOption> soList =
DBControl.retrieveStockOptionsByOS(currentDayOs.get(i).getId());

                    if(!soList.isEmpty()){
                         //we assume that all the valid SOs of the OS are eligible for
transfer the next day
```

```
                    for(int k=0; k<soList.size(); k++)

     currentDayOs.get(i).setStockOptionsTransferred(currentDayOs.get(i).getStockO
ptionsTransferred()+soList.get(k).getSoNumber());
                }
                    logger.info("OS #"+currentDayOs.get(i).getId()+" can
potentially sell "+currentDayOs.get(i).getStockOptionsTransferred() +" Stock
Options in SM.");

                DBControl.updateOriginServer(currentDayOs.get(i), day);
            }
        }
          return currentDayOs;
     }

     /*
      * This method is used to perform the final calculations before the day ends
for all the OSs
      */
     public static ArrayList<OriginServer> concludeData(ArrayList<OriginServer>
currentDayOs,int day){

          for(int i=0; i<currentDayOs.size(); i++){

               //if the OS was a seller
               if(currentDayOs.get(i).getOsAction()==1){

                    //we calculate the SOs to be transferred the next day

     currentDayOs.get(i).setStockOptionsTransferred(currentDayOs.get(i).getStockO
ptionsTransferred()-currentDayOs.get(i).getExchangedSO());
                    //we calculate the traffic that will be bought from CDN with
```

```
penalty price
                    currentDayOs.get(i).setCdnTrafficWithPenalty(0);

                    //we calculate the traffic to be transferred the next day
(not the SOs)

     currentDayOs.get(i).setTrafficTransfer(currentDayOs.get(i).getInitialUnusedT
raffic()-currentDayOs.get(i).getExchangedBW());

                    if(currentDayOs.get(i).getTrafficTransfer()<0){
                         currentDayOs.get(i).setTrafficTransfer(0);
                    }

                    currentDayOs.get(i).setRemainingTrafficAfterAuction(0);

               }//if the OS was a buyer
               else if(currentDayOs.get(i).getOsAction()==-1){

                    //we calculate the SOs to be transferred the next day
                    currentDayOs.get(i).setStockOptionsTransferred(0);
                    //we calculate the traffic that will be bought from CDN with
penalty price

     currentDayOs.get(i).setCdnTrafficWithPenalty(currentDayOs.get(i).getRemainin
gTrafficAfterAuction()*(-1));

                    //we calculate the traffic to be transferred the next day
(not the SOs)

     currentDayOs.get(i).setTrafficTransfer(currentDayOs.get(i).getInitialUnusedT
raffic()-currentDayOs.get(i).getExchangedBW());

                    if(currentDayOs.get(i).getTrafficTransfer()<0){
```

```
                        currentDayOs.get(i).setTrafficTransfer(0);
                    }

                    //if the OS buys traffic with penalty we update the cdn
wallet

                     if(currentDayOs.get(i).getCdnTrafficWithPenalty()>0){
                              CdnWallet cdnWallet = new CdnWallet();
                              cdnWallet.setDay(day);
                              cdnWallet.setOsId(currentDayOs.get(i).getId());

     cdnWallet.setPrice(currentDayOs.get(i).getHighPrice());
                              cdnWallet.setPriceFlag("H");

     cdnWallet.setResources(currentDayOs.get(i).getCdnTrafficWithPenalty());
                              cdnWallet.setResourceType("BW");
                              cdnWallet.setSource("SM");

     cdnWallet.setWallet(Precision.round(currentDayOs.get(i).getHighPrice()*curre
ntDayOs.get(i).getCdnTrafficWithPenalty(),5));

                              //we update the CDN Wallet after the purchase of
the BW
                              DBControl.insertCDNWallet(cdnWallet);
                     }
               }

               //we update the data in the DB
               DBControl.updateOriginServer(currentDayOs.get(i), day);
          }

          //we update all the SOs by subtracting 1 day from from the days till
```

```
expiration
            //and we print the current state of the OS at the end of day
            for(int i=0; i<currentDayOs.size(); i++){

                  int soTransferred = 0;

                  logger.info(currentDayOs.get(i).toString());

                  int osId  = currentDayOs.get(i).getId();

                  //we retrieve all the valid SOs of the OS
                  ArrayList<StockOption> soList =
DBControl.retrieveStockOptionsByOS(osId);

                  if(!soList.isEmpty()){

                        for(int j = 0; j<soList.size(); j++){
                              StockOption so = soList.get(j);
                              so.setDaysExpire(so.getDaysExpire()-1);
                              DBControl.updateStockOptions(so);
                        }

                  }

                  //we retrieve all the valid SOs of the OS to calculate how many
will be transferred the next day
                   soList = DBControl.retrieveStockOptionsByOS(osId);

                  if(!soList.isEmpty()){
                        for(int j = 0; j<soList.size(); j++){
                              StockOption so = soList.get(j);
```

```
                        soTransferred += so.getSoNumber();
                    }
                }

                currentDayOs.get(i).setStockOptionsTransferred(soTransferred);

                DBControl.updateOriginServer(currentDayOs.get(i), day);
            }

            return currentDayOs;
        }
```

- **performAuction (for Resources)**

The below function is responsible for the simulation of the auction procedure of resources in the SM.

```
    /*
     * This method will be responsible to perform the auction of BW in the SM
     */
    @SuppressWarnings("unchecked")
    public static ArrayList<OriginServer> performAuction(ArrayList<OriginServer>
currentDayOs,int day){

            //we initialize the variables for the buyers/sellers of BW in SM

            ArrayList<OriginServer> buyersList = new ArrayList<OriginServer>();
            ArrayList<OriginServer> sellersList = new ArrayList<OriginServer>();

            ArrayList<OriginServer> osList = new ArrayList<OriginServer>();

            //we identify the potential buyers/sellers for the auction in the
```

```
Secondary Market
                for(int i=0; i<currentDayOs.size(); i++){


                    int initialUnusedTraffic = (-
1)*currentDayOs.get(i).getInitialUnusedTraffic();



                    if(initialUnusedTraffic>0){


                        currentDayOs.get(i).setOsAction(-1); // we set the action
of the OS as a buyer


                         // we set the initial value for the Traffic AFTER the
auction


     currentDayOs.get(i).setRemainingTrafficAfterAuction(currentDayOs.get(i).getInit
ialUnusedTraffic());


                        currentDayOs.get(i).setSmSellingPriceBW(0); // we set the
sm selling price to 0 since the OS is not a seller


                        DBControl.updateOriginServer(currentDayOs.get(i), day); //
we update the data in the DB


                        buyersList.add(currentDayOs.get(i)); // we add the OS to
the potential buyers
                    }
                   else if(initialUnusedTraffic<0 ){


                    currentDayOs.get(i).setOsAction(1);  // we set the action of the
OS as a seller


                    // we set the initial value for the Traffic AFTER the auction
```

```
     currentDayOs.get(i).setRemainingTrafficAfterAuction(currentDayOs.get(i).getInit
ialUnusedTraffic());


                    //we calculate the selling price of the BW in the SM - We set it
as the low_price

     currentDayOs.get(i).setSmSellingPriceBW(currentDayOs.get(i).getLowPrice());


                    DBControl.updateOriginServer(currentDayOs.get(i), day); // we
update the data in the DB
                         sellersList.add(currentDayOs.get(i)); // we add the OS to
the potential sellers
                    }
               }

               Logger.info("Number of Potential
{Buyers,Sellers}:"+"{"+buyersList.size()+","+sellersList.size()+"}");

               //we check if we have at least 1 buyer and 1 seller
               if(!buyersList.isEmpty() && !sellersList.isEmpty()){

                    //we sort the sellers based on their selling price in ascending
order low->max

                    Collections.sort(sellersList);



                    for(int j=0; j < sellersList.size(); j++){ // we find the
potential buyers for the respective seller

                         double smSellingPriceBW =
sellersList.get(j).getSmSellingPriceBW();
```

```
                        int potentialBuyers = 0 ;

                        for(int i=0; i<buyersList.size(); i++){

                            /*
                             * The penalty price that the specific buyer has from
CDN is the maximum price based on which the OS can buy from SM
                             *
                             */
                            //High Penalty Version
                            if(buyersList.get(i).getHighPrice()>smSellingPriceBW
&& (-1)*buyersList.get(i).getRemainingTrafficAfterAuction()>0){
                                potentialBuyers++;
                            }

                            //Low penalty Version
//                            if(buyersList.get(i).getLowPrice()>smSellingPriceBW &&
(-1)*buyersList.get(i).getRemainingTrafficAfterAuction()>0){
//                                potentialBuyers++;
//                            }

                        }

                        if(potentialBuyers>0){

                            Logger.info("The buyers for Seller
{"+sellersList.get(j).getId()+"} are "+potentialBuyers);

                            //we proceed with the buy splitted fairly between all
buyers
```

```
                                //we keep the total number of BW that the Seller is
selling
                                int totalSellingBW =
sellersList.get(j).getRemainingTrafficAfterAuction();


                                //we keep the total number of BW that need to be
bought
                                int totalBuyingBW = 0;


                                for(int k=0; k<buyersList.size(); k++){
                                     totalBuyingBW +=
buyersList.get(k).getRemainingTrafficAfterAuction();
                                }


                                totalBuyingBW = totalBuyingBW*(-1);


                                Logger.info("BW to be sold "+totalSellingBW+" and
"+totalBuyingBW + " BW need to be bought in SM.");


                                if(totalSellingBW>=totalBuyingBW && totalBuyingBW>0){


                                     Logger.info("The amount of BW to be sold is more
(or the same) than the amount "
                                               + "needed to be bought. The
"+buyersList.size()+" buyers will buy all the BW they need from seller
{"+sellersList.get(j).getId()+"}."
                                               + " Seller
{"+sellersList.get(j).getId()+"} will remain with "+ (totalSellingBW-totalBuyingBW) +
" in BW.");


                                     for(int i = 0; i < buyersList.size(); i++)
                                          makeSale(sellersList.get(j),
buyersList.get(i), day,0);
```

```
                                    break; // we exit the loop the rest of the
sellers(if any) will not sell their BW

                                }else if (totalSellingBW<totalBuyingBW){

                                    double sellingPercentage =
(double)totalSellingBW/totalBuyingBW;

                                    sellingPercentage =
Precision.round(sellingPercentage, 15);

                                    Logger.info("The amount of BW to be sold is less
than the amount needed to be bought. The "+buyersList.size()+" buyers "
                                            + "will buy "+
Precision.round(sellingPercentage*100,15) + "% of their needs (BW) from Seller
{"+sellersList.get(j).getId()+"} . The Seller will not have any BW remaining after
SM.");

                                    Logger.info("Selling Percentage of
BW:"+sellingPercentage);

                                    for(int i = 0; i < buyersList.size(); i++){

                                        int buyersNeeds =
buyersList.get(i).getRemainingTrafficAfterAuction()*(-1);

                                        int partialBought = (int)
Math.round(sellingPercentage*buyersNeeds);
                                        if(partialBought==0){
                                            partialBought++;
                                        }
```

```
                                        Logger.info("Buyer
{"+buyersList.get(i).getId()+"} will buy:"+partialBought);


                                        makeSale(sellersList.get(j),
buyersList.get(i), day,partialBought);


                                        //we set the remaining SOs after auction
for the seller to 0
     sellersList.get(j).setRemainingTrafficAfterAuction(0);

     DBControl.updateOriginServer(sellersList.get(j), day);
                                    }
                               }
                          }
                     }
                }else
                  Logger.info("Not enough Buyers or/and Sellers in the Secondary
Market to perform the auction.");


                //here we return the list with the updated data after the auction of
the day
                if(sellersList.size()!=0){
                     for(int i = 0; i<sellersList.size(); i++)
                          osList.add(sellersList.get(i));
                }


                if(buyersList.size()!=0){
                     for(int i = 0; i<buyersList.size(); i++)
                          osList.add(buyersList.get(i));
                }


                for(int i=0; i<osList.size(); i++)
```

```
                    logger.info(osList.get(i).toString());


                return osList;

            }
```

- **makeSale (for Resources)**

This function is responsible for performing the sale of Resources in the SM between the Origin Servers.

```
    /*
           * This function is responsible to perform the sale
           */
          private static void makeSale(OriginServer seller,OriginServer buyer,int
day, int partial){


               if(partial>0){


                    //update the remaining traffic after auction for buyer and
seller


    seller.setRemainingTrafficAfterAuction(seller.getRemainingTrafficAfterAuction()
- partial);


    buyer.setRemainingTrafficAfterAuction(buyer.getRemainingTrafficAfterAuction() +
partial);


                    //update the wallet for the OSs
                    seller.setWalletChange(seller.getWalletChange() +
partial*seller.getSmSellingPriceBW());
                    buyer.setWalletChange(buyer.getWalletChange()  -
partial*seller.getSmSellingPriceBW());


                    //we update the bw_exchanged for buyer and seller
```

```
                    seller.setExchangedBW(seller.getExchangedBW()+partial);
                    buyer.setExchangedBW(buyer.getExchangedBW()+partial);

                }else{

                    int buyerTrafficNeeds =
buyer.getRemainingTrafficAfterAuction()*(-1);

                    //update the remaining traffic after auction for buyer and
seller

    seller.setRemainingTrafficAfterAuction(seller.getRemainingTrafficAfterAuction()
- buyerTrafficNeeds);

    buyer.setRemainingTrafficAfterAuction(buyer.getRemainingTrafficAfterAuction() +
buyerTrafficNeeds);

                    //update the wallet for the OSs
                    seller.setWalletChange(seller.getWalletChange() +
buyerTrafficNeeds*seller.getSmSellingPriceBW());
                    buyer.setWalletChange(buyer.getWalletChange()  -
buyerTrafficNeeds*seller.getSmSellingPriceBW());

                    //we update the bw_exchanged for buyer and seller

    seller.setExchangedBW(seller.getExchangedBW()+buyerTrafficNeeds);
                    buyer.setExchangedBW(buyer.getExchangedBW()+buyerTrafficNeeds);

                }
                //update also the DB data
                DBControl.updateOriginServer(seller, day);
                DBControl.updateOriginServer(buyer, day);
            }
```

- **concludeData**

This function is responsible for the conclusion of data at the end of the day for all Origin Servers.

```
          /*
           * This method is used to perform the final calculations before the day
ends for all the OSs
           */
          public static ArrayList<OriginServer> concludeData(ArrayList<OriginServer>
currentDayOs,int day){

               for(int i=0; i<currentDayOs.size(); i++){

                    //we calculate the BW to be bought from CDN with the high
penalty price
                    if(currentDayOs.get(i).getRemainingTrafficAfterAuction()<0)

     currentDayOs.get(i).setCdnTrafficWithPenalty(currentDayOs.get(i).getRemainingTr
afficAfterAuction()*(-1));
                    else
                         currentDayOs.get(i).setCdnTrafficWithPenalty(0);

                    //we calculate the BW to be transferred the next day
                    if(currentDayOs.get(i).getRemainingTrafficAfterAuction()>0)

     currentDayOs.get(i).setTrafficTransfer(currentDayOs.get(i).getRemainingTrafficA
fterAuction());
                    else
                         currentDayOs.get(i).setTrafficTransfer(0);

                    //we update the data in the DB
                    DBControl.updateOriginServer(currentDayOs.get(i), day);
```

```
                    //if the OS buys traffic with penalty we update the cdn wallet

                     if(currentDayOs.get(i).getCdnTrafficWithPenalty()>0){
                              CdnWallet cdnWallet = new CdnWallet();
                              cdnWallet.setDay(day);
                              cdnWallet.setOsId(currentDayOs.get(i).getId());

     cdnWallet.setPrice(currentDayOs.get(i).getHighPrice());
                              cdnWallet.setPriceFlag("H");

     cdnWallet.setResources(currentDayOs.get(i).getCdnTrafficWithPenalty());
                              cdnWallet.setResourceType("BW");
                              cdnWallet.setSource("SM");

     cdnWallet.setWallet(Precision.round(currentDayOs.get(i).getHighPrice()*currentD
ayOs.get(i).getCdnTrafficWithPenalty(),5));

                              //we update the CDN Wallet after the purchase of the
BW
                              DBControl.insertCDNWallet(cdnWallet);
                     }

               }

               //we update all the SOs by subtracting 1 day from from the days till
expiration
               //and we print the current state of the OS at the end of day
               for(int i=0; i<currentDayOs.size(); i++){

                    int soTransferred = 0;
```

```
                    logger.info(currentDayOs.get(i).toString());


                    int osId  = currentDayOs.get(i).getId();


                    //we retrieve all the valid SOs of the OS and subtract the
remaining days till expire
                    ArrayList<StockOption> soList =
DBControl.retrieveStockOptionsByOS(osId);


                    if(!soList.isEmpty()){
                          for(int j = 0; j<soList.size(); j++){
                                StockOption so = soList.get(j);
                                so.setDaysExpire(so.getDaysExpire()-1);
                                DBControl.updateStockOptions(so);
                          }
                    }


                    //we retrieve all the valid SOs of the OS to calculate how many
will be transferred the next day
                     soList = DBControl.retrieveStockOptionsByOS(osId);


                    if(!soList.isEmpty()){
                          for(int j = 0; j<soList.size(); j++){
                                StockOption so = soList.get(j);
                                soTransferred += so.getSoNumber();
                          }
                    }


                    currentDayOs.get(i).setStockOptionsTransferred(soTransferred);


                    DBControl.updateOriginServer(currentDayOs.get(i), day);
```

```
                }

                return currentDayOs;

            }
```

ANNEX III: CDN PRICING TABLE

The table below refers to the prices of Stock Options for each cost plan provided by the CDN. The price of the stock options is based on the low price of each cost plan.

Table 20: CDN Pricing Table used in the Simulation

CDN						
Cost Plan	Normal Price	High Price	Low Price	SO Price	SO Duration	Plan Description
P1	0.035	0.042	0.033	0.00200452	1	0GB-10GB
P1	0.035	0.042	0.033	0.00200904	2	0GB-10GB
P1	0.035	0.042	0.033	0.002013559	3	0GB-10GB
P1	0.035	0.042	0.033	0.002018077	4	0GB-10GB
P1	0.035	0.042	0.033	0.002022595	5	0GB-10GB
P1	0.035	0.042	0.033	0.002027112	6	0GB-10GB
P1	0.035	0.042	0.033	0.002031629	7	0GB-10GB
P1	0.035	0.042	0.033	0.002036145	8	0GB-10GB
P1	0.035	0.042	0.033	0.00204066	9	0GB-10GB
P1	0.035	0.042	0.033	0.002045175	10	0GB-10GB
P1	0.035	0.042	0.033	0.002049689	11	0GB-10GB
P1	0.035	0.042	0.033	0.002054202	12	0GB-10GB
P1	0.035	0.042	0.033	0.002058715	13	0GB-10GB
P1	0.035	0.042	0.033	0.002063227	14	0GB-10GB
P1	0.035	0.042	0.033	0.002067739	15	0GB-10GB
P1	0.035	0.042	0.033	0.00207225	16	0GB-10GB
P1	0.035	0.042	0.033	0.00207676	17	0GB-10GB
P1	0.035	0.042	0.033	0.00208127	18	0GB-10GB
P1	0.035	0.042	0.033	0.002085779	19	0GB-10GB
P1	0.035	0.042	0.033	0.002090287	20	0GB-10GB
P1	0.035	0.042	0.033	0.002094795	21	0GB-10GB
P1	0.035	0.042	0.033	0.002099302	22	0GB-10GB
P1	0.035	0.042	0.033	0.002103809	23	0GB-10GB
P1	0.035	0.042	0.033	0.002108315	24	0GB-10GB
P1	0.035	0.042	0.033	0.00211282	25	0GB-10GB
P1	0.035	0.042	0.033	0.002117325	26	0GB-10GB
P1	0.035	0.042	0.033	0.002121829	27	0GB-10GB
P1	0.035	0.042	0.033	0.002126333	28	0GB-10GB
P1	0.035	0.042	0.033	0.002130836	29	0GB-10GB
P1	0.035	0.042	0.033	0.002135338	30	0GB-10GB
P2	0.03	0.036	0.028	0.002003835	1	10GB-100GB
P2	0.03	0.036	0.028	0.00200767	2	10GB-100GB
P2	0.03	0.036	0.028	0.002011504	3	10GB-100GB
P2	0.03	0.036	0.028	0.002015338	4	10GB-100GB
P2	0.03	0.036	0.028	0.002019172	5	10GB-100GB
P2	0.03	0.036	0.028	0.002023004	6	10GB-100GB
P2	0.03	0.036	0.028	0.002026836	7	10GB-100GB
P2	0.03	0.036	0.028	0.002030668	8	10GB-100GB
P2	0.03	0.036	0.028	0.002034499	9	10GB-100GB

P2	0.03	0.036	0.028	0.00203833	10	10GB-100GB
P2	0.03	0.036	0.028	0.00204216	11	10GB-100GB
P2	0.03	0.036	0.028	0.00204599	12	10GB-100GB
P2	0.03	0.036	0.028	0.002049819	13	10GB-100GB
P2	0.03	0.036	0.028	0.002053647	14	10GB-100GB
P2	0.03	0.036	0.028	0.002057475	15	10GB-100GB
P2	0.03	0.036	0.028	0.002061303	16	10GB-100GB
P2	0.03	0.036	0.028	0.00206513	17	10GB-100GB
P2	0.03	0.036	0.028	0.002068956	18	10GB-100GB
P2	0.03	0.036	0.028	0.002072782	19	10GB-100GB
P2	0.03	0.036	0.028	0.002076607	20	10GB-100GB
P2	0.03	0.036	0.028	0.002080432	21	10GB-100GB
P2	0.03	0.036	0.028	0.002084257	22	10GB-100GB
P2	0.03	0.036	0.028	0.00208808	23	10GB-100GB
P2	0.03	0.036	0.028	0.002091904	24	10GB-100GB
P2	0.03	0.036	0.028	0.002095726	25	10GB-100GB
P2	0.03	0.036	0.028	0.002099549	26	10GB-100GB
P2	0.03	0.036	0.028	0.00210337	27	10GB-100GB
P2	0.03	0.036	0.028	0.002107192	28	10GB-100GB
P2	0.03	0.036	0.028	0.002111012	29	10GB-100GB
P2	0.03	0.036	0.028	0.002114832	30	10GB-100GB
P3	0.025	0.03	0.023	0.00200315	1	100GB-1000GB
P3	0.025	0.03	0.023	0.002006301	2	100GB-1000GB
P3	0.025	0.03	0.023	0.00200945	3	100GB-1000GB
P3	0.025	0.03	0.023	0.002012599	4	100GB-1000GB
P3	0.025	0.03	0.023	0.002015748	5	100GB-1000GB
P3	0.025	0.03	0.023	0.002018896	6	100GB-1000GB
P3	0.025	0.03	0.023	0.002022044	7	100GB-1000GB
P3	0.025	0.03	0.023	0.002025192	8	100GB-1000GB
P3	0.025	0.03	0.023	0.002028339	9	100GB-1000GB
P3	0.025	0.03	0.023	0.002031485	10	100GB-1000GB
P3	0.025	0.03	0.023	0.002034631	11	100GB-1000GB
P3	0.025	0.03	0.023	0.002037777	12	100GB-1000GB
P3	0.025	0.03	0.023	0.002040922	13	100GB-1000GB
P3	0.025	0.03	0.023	0.002044067	14	100GB-1000GB
P3	0.025	0.03	0.023	0.002047212	15	100GB-1000GB
P3	0.025	0.03	0.023	0.002050356	16	100GB-1000GB
P3	0.025	0.03	0.023	0.002053499	17	100GB-1000GB
P3	0.025	0.03	0.023	0.002056642	18	100GB-1000GB
P3	0.025	0.03	0.023	0.002059785	19	100GB-1000GB
P3	0.025	0.03	0.023	0.002062927	20	100GB-1000GB
P3	0.025	0.03	0.023	0.002066069	21	100GB-1000GB
P3	0.025	0.03	0.023	0.002069211	22	100GB-1000GB
P3	0.025	0.03	0.023	0.002072352	23	100GB-1000GB
P3	0.025	0.03	0.023	0.002075492	24	100GB-1000GB
P3	0.025	0.03	0.023	0.002078632	25	100GB-1000GB

P3	0.025	0.03	0.023	0.002081772	26	100GB-1000GB
P3	0.025	0.03	0.023	0.002084911	27	100GB-1000GB
P3	0.025	0.03	0.023	0.00208805	28	100GB-1000GB
P3	0.025	0.03	0.023	0.002091189	29	100GB-1000GB
P3	0.025	0.03	0.023	0.002094327	30	100GB-1000GB
P4	0.02	0.024	0.019	0.001002603	1	1000GB-10000GB
P4	0.02	0.024	0.019	0.001005205	2	1000GB-10000GB
P4	0.02	0.024	0.019	0.001007807	3	1000GB-10000GB
P4	0.02	0.024	0.019	0.001010408	4	1000GB-10000GB
P4	0.02	0.024	0.019	0.001013009	5	1000GB-10000GB
P4	0.02	0.024	0.019	0.00101561	6	1000GB-10000GB
P4	0.02	0.024	0.019	0.00101821	7	1000GB-10000GB
P4	0.02	0.024	0.019	0.001020811	8	1000GB-10000GB
P4	0.02	0.024	0.019	0.00102341	9	1000GB-10000GB
P4	0.02	0.024	0.019	0.00102601	10	1000GB-10000GB
P4	0.02	0.024	0.019	0.001028609	11	1000GB-10000GB
P4	0.02	0.024	0.019	0.001031207	12	1000GB-10000GB
P4	0.02	0.024	0.019	0.001033806	13	1000GB-10000GB
P4	0.02	0.024	0.019	0.001036403	14	1000GB-10000GB
P4	0.02	0.024	0.019	0.001039001	15	1000GB-10000GB
P4	0.02	0.024	0.019	0.001041598	16	1000GB-10000GB
P4	0.02	0.024	0.019	0.001044195	17	1000GB-10000GB
P4	0.02	0.024	0.019	0.001046792	18	1000GB-10000GB
P4	0.02	0.024	0.019	0.001049388	19	1000GB-10000GB
P4	0.02	0.024	0.019	0.001051984	20	1000GB-10000GB
P4	0.02	0.024	0.019	0.001054579	21	1000GB-10000GB
P4	0.02	0.024	0.019	0.001057174	22	1000GB-10000GB
P4	0.02	0.024	0.019	0.001059769	23	1000GB-10000GB
P4	0.02	0.024	0.019	0.001062363	24	1000GB-10000GB
P4	0.02	0.024	0.019	0.001064957	25	1000GB-10000GB
P4	0.02	0.024	0.019	0.001067551	26	1000GB-10000GB
P4	0.02	0.024	0.019	0.001070144	27	1000GB-10000GB
P4	0.02	0.024	0.019	0.001072737	28	1000GB-10000GB
P4	0.02	0.024	0.019	0.00107533	29	1000GB-10000GB
P4	0.02	0.024	0.019	0.001077922	30	1000GB-10000GB
P5	0.015	0.018	0.014	0.001001918	1	10000GB-100000GB
P5	0.015	0.018	0.014	0.001003835	2	10000GB-100000GB
P5	0.015	0.018	0.014	0.001005752	3	10000GB-100000GB
P5	0.015	0.018	0.014	0.001007669	4	10000GB-100000GB
P5	0.015	0.018	0.014	0.001009586	5	10000GB-100000GB
P5	0.015	0.018	0.014	0.001011502	6	10000GB-100000GB
P5	0.015	0.018	0.014	0.001013418	7	10000GB-100000GB
P5	0.015	0.018	0.014	0.001015334	8	10000GB-100000GB
P5	0.015	0.018	0.014	0.00101725	9	10000GB-100000GB
P5	0.015	0.018	0.014	0.001019165	10	10000GB-100000GB
P5	0.015	0.018	0.014	0.00102108	11	10000GB-100000GB

P5	0.015	0.018	0.014	0.001022995	12	10000GB-100000GB
P5	0.015	0.018	0.014	0.001024909	13	10000GB-100000GB
P5	0.015	0.018	0.014	0.001026824	14	10000GB-100000GB
P5	0.015	0.018	0.014	0.001028738	15	10000GB-100000GB
P5	0.015	0.018	0.014	0.001030651	16	10000GB-100000GB
P5	0.015	0.018	0.014	0.001032565	17	10000GB-100000GB
P5	0.015	0.018	0.014	0.001034478	18	10000GB-100000GB
P5	0.015	0.018	0.014	0.001036391	19	10000GB-100000GB
P5	0.015	0.018	0.014	0.001038304	20	10000GB-100000GB
P5	0.015	0.018	0.014	0.001040216	21	10000GB-100000GB
P5	0.015	0.018	0.014	0.001042128	22	10000GB-100000GB
P5	0.015	0.018	0.014	0.00104404	23	10000GB-100000GB
P5	0.015	0.018	0.014	0.001045952	24	10000GB-100000GB
P5	0.015	0.018	0.014	0.001047863	25	10000GB-100000GB
P5	0.015	0.018	0.014	0.001049774	26	10000GB-100000GB
P5	0.015	0.018	0.014	0.001051685	27	10000GB-100000GB
P5	0.015	0.018	0.014	0.001053596	28	10000GB-100000GB
P5	0.015	0.018	0.014	0.001055506	29	10000GB-100000GB
P5	0.015	0.018	0.014	0.001057416	30	10000GB-100000GB
P6	0.012	0.014	0.011	0.001001507	1	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001003013	2	100000GB-1000000GB
P6	0.012	0.014	0.011	0.00100452	3	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001006026	4	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001007532	5	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001009037	6	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001010543	7	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001012048	8	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001013553	9	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001015058	10	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001016563	11	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001018067	12	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001019572	13	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001021076	14	100000GB-1000000GB
P6	0.012	0.014	0.011	0.00102258	15	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001024083	16	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001025587	17	100000GB-1000000GB
P6	0.012	0.014	0.011	0.00102709	18	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001028593	19	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001030096	20	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001031598	21	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001033101	22	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001034603	23	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001036105	24	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001037607	25	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001039108	26	100000GB-1000000GB
P6	0.012	0.014	0.011	0.00104061	27	100000GB-1000000GB

P6	0.012	0.014	0.011	0.001042111	28	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001043612	29	100000GB-1000000GB
P6	0.012	0.014	0.011	0.001045113	30	100000GB-1000000GB

REFERENCES

[1] E. Vathias, E. Katsarou, S. Hadjiefthymiades, A Secondary Market Metaphor for Content Distribution Networks, National and Kapodistrian University of Athens, Greece.
[2] E. Vathias, S. Hadjiefthymiades, A Stock Options Metaphor for Content Delivery Networks, National and Kapodistrian University of Athens.
[3] Content Delivery Networks. https://www.incapsula.com/cdn-guide/what-is-cdn-how-it-works.html, Incapsula.com [Accessed on 27/10/2018].
[4] https://www.imperva.com/learn/performance/what-is-cdn-how-it-works/ , Imperva.com [Accessed on 27/10/2018].
[5] Reverse Cache Proxy, Microsoft Documentation https://msdn.microsoft.com/en-us/library/windows/desktop/dd892097(v=vs.85).aspx, Microsoft.com [Accessed on 27/10/2018].
[6] https://www.cloudflare.com/learning/cdn/glossary/anycast-network/, Cloudflare.com [Accessed on 27/10/2018].
[7] E. Vathias, D. Nikolopoulos, S. Hadjiefthymiades, A Capital Market Metaphor for Content Delivery Network Resources, National and Kapodistrian University of Athens, Greece.
[8] John C. Hull, Options, Futures and Other Derivatives 8th Edition, Prentice Hall, 2012, pp. 194-233.
[9] Κωνσταντίνος Μ. Αντωνίου, Προδραστική Διαχείριση Πόρων σε ασύρματα δίκτυα με τη χρήση δικαιωμάτων προαίρεσης (stock options) , Διπλωματική Εργασία ΕΚΠΑ, Αθήνα, 2008 [in Greek].
[10] https://cloud.google.com/storage/docs/storage-classes , Google Cloud [Accessed on 15/01/2019]
[11] https://cloud.google.com/storage/pricing#pricing-example-detailed , Google Cloud [Accessed on 15/01/2019]