GEZORA: A SELF-SCALABLE CONTENT DISTRIBUTION NETWORK

Masterarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Milan Nikolic
2010

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Abstract

Content Distribution Networks (CDNs) are cooperative server environments, which assist popular web sites in being resilient against so-called flash crowds - instantaneous traffic spikes, generated by concurrent user accesses to the same web content. Today's CDNs have substantially improved the content delivery. Still many of them have disadvantages regarding the adaptivity to sudden and fast growing client request rates.

In this Master thesis, we designed a novel CDN, codenamed Gezora [1], which addresses common CDNs issues, such as: the unpredictability of client requests, self-adaptivity to the fast request frequency changes, robustness, and on-demand scalability. Using a request frequency monitoring system, based on the exponential weighted moving average, feedback control loop [2], random early redirection algorithm [3] and a URL-based content clustering, Gezora establishes a hybrid self-reactive CDN, which is capable of changing server's role dynamically for every content subset, from the normal content provider to the pure load balancer (reverse proxy) and achieves the load balancing while distributing the web content in the same time. The main requirement for Gezora is to avoid server overloading and to minimize the response time at the peak client request rates. The main advantage over other CDNs, is the ability to change the server role dynamically at the run-time and the self-adaptive request routing mechanism which is able to achieve fast on-demand scaling without applying any predictive models or Markov chains [4].

The current Gezora prototype is implemented using Java-based Apache Tomcat web server, J2EE technology and Java servlets, which are the Java classes whose instances accept and answer client requests within a Java based web server. The evaluation of Gezora is accomplished using web server stress tests. The evaluation results show significant improvements regarding the server response times during the peak request rates. More specifically, the test scenario applied on the Gezora prototype, outperformed a pure single-server approach with a 30% lower average request time.

Acknowledgment

I would like to thank Prof. Dr. Torsten Braun for giving me the opportunity to do my Master thesis in his research group. During the course of my thesis, he was always very patient and supportive and for this I am very grateful.

I am also very grateful to Dr. Dragan Milic for giving me an in-depth insight into the technical aspects of PlanetLab and J2EE. Dragan's innovative ideas for the J2EE implementation have helped steer my thesis in the right direction and I really appreciate it.

I would like to thank all members of Prof. Braun's research group for the friendly working environment and for making my master's thesis a memorable experience.

I would like to express my genuine gratitude to Philipp Hurni, for correcting my thesis and for his invaluable tips about scientific writing. You are a true friend.

Special thanks go to my sister, Milica Nikolic, for the final spelling revision of my thesis and her constant support.

Last but not least, I would like to express my utter gratitude to my parents, for their unconditional love and constant support. I couldn't have done this without you.

Milan Nikolic, 2010

Contents

List of Figures

Listings

Chapter 1

Introduction and Motivation

1.1 The Emergence of Content Distribution Systems

Since the Internet has been established as a key technology in our everyday life, traffic has been constantly increasing, enhanced by the acceptance of broadband access. Content has become much more complex, and user traffic variable and unpredictable. The ever evolving nature of the Internet presents new challenges in content management and delivery. For example, many of the most popular web services suffer from congestion due to the heavy demands made upon their services. A sudden increase, or spike, in the demand for web content can often cause an overload on particular web servers and, as a result, a performance bottleneck can arise, causing high server response times. Such unexpected peaks in the demand cause significant strain on a web server. Subsequently, many web servers become overwhelmed with a sudden increase in traffic, and the host web site content becomes temporarily unavailable.

Researchers are becoming more and more aware of this fact. Most content providers view the Internet as a key vehicle in their aim to bring goods to end users. Therefore, a decrease in the quality of the service due to high levels of demand which results in longer download times will leave the end users frustrated. Modern companies expect significant financial revenues from web based e-business. As a result, in the past few years, an evolution of technologies aimed at improving content delivery and service provision over the web has been seen. A combination of structures and the supporting technologies form a new type of network, which is often referred to as a Content Network (CN). Single web servers evolved to a complex cooperative environments, which are more resilient to the sudden traffic changes. The main priority of a number of CNs is to address the levels of performance and to minimize problems and to maintain the Quality of Service (QoS) by employing a variety of mechanisms. One such approach is to modify the network infrastructure, e.g. by improving the web server hardware by adding a high-speed processor, more memory, disk space, or even a multi-processor system. However, this approach is not an optimal solution, since the small enhancements are generally not viable for a long time period, which results in complete server infrastructure needing to be replaced.

1.2 CDN - Technological Improvement for new Internet Trends

With the development of the Internet the end users have changed too. Today's users are becoming more and more content providers while they are primarily content consumers. The main reason for this trend is, that the Internet today consists in more content publishing using web platforms such as Youtube [5] and Facebook [6], or even content sharing using Peer to Peer (P2P) networks, rather than a classical server-client usage. As the technical infrastructure increases both in physical terms and by capacity, it is becoming easier for the private users to maintain their own permanent Internet node on a broadband connection and run various types of servers such as e-mail, file transfer, web content, remote login, etc. The usual option for users or small organizations who wish to maintain a web site is to place it under the regime of a commercial web hosting service. This approach is functioning well under low load. In case of high demands, web hosting companies additionally utilize the Content Distribution Networks (CDN) [7, 8, 9]. A CDN is a group of globally distributed servers networked together across the Internet to cooperate transparently, in order to assure fast and reliable content-delivery to end users. A CDN also uses smart request routing mechanisms, which help today's web sites to be resilient against flash crowds: quick increases of user requests that lead to server overloading situations. Some of web hosting services use also caching proxies or server farms as alternative solution for pushing the popular content closer to the end users.

1.3 Limitations of Existing CDNs

Today's large server farms and hierarchical caching mechanisms are useful techniques to address Internet web performance problems, they have a variety of limitations. First, as the servers are deployed close to the origin server, they actually do very little to improve the network performance when congestion occurs. Furthermore, although caching proxies can be beneficial, they cache objects based on the client's demands. This often forces the providers of the most popular content to invest in large server farms, offering load balancing and high bandwidth connections, so as to maintain performance to keep up with the demand. In order to address the existing content distribution limitations, CDNs were deployed in the late 1990s. Naturally, the introduction of CDNs meant, that content providers began to put their web sites onto CDN, as they quickly realized the benefits of increased reliability and scalability without the need to maintain an expensive infrastructure, and consequently, several initiatives coincided with the development of CDNs. Still, today's CDNs have a variety of issues considering the adaptivity on fast request number changes and scaling. Currently, medium to large-scale CDNs suffer from such issues. The size and amount of provided data grows every year, so the servers providing it must grow accordingly. Some hosts, like *sourceforge.net*, provide data in excess of one terabyte. The biggest and most often used mirror server in Switzerland, the SWITCH mirror service, has passed 3.6 terabyte data and is still growing. The mirrors for the Linux OS are mostly provided by universities and large Internet Service Providers (ISPs). Obviously, the bandwidth must grow at the same rate, so the network connections to web servers can become bottlenecks due to large number of client requests within a short time interval. This situation implies the server scalability issues, since a single server can not meet the client requirements during the high load rate.

Popular web sites get easily overloaded with the huge number of user requests. With the increase of the number of user requests, either the server's processing capacity (CPU, RAM) or the available bandwidth (Internet Connection) are exceeded. If such a situation occurs, user requests are usually dropped. This results in increased access delay or even unavailability of the server. Scalability issues become even more severe when sudden or unique events occur, that are of big interest to the public, such as breaking news or new software (updates/patches). The easiest but not optimal way to increase the scalability of CDNs is to increase the amount and capacity of servers (e.g. server farms). One example of such services are the Amazon Elastic Cloud services with it's innovative cloud auto-scaling service, called Scalr [10], which uses a number of elastic load balancers to handle spikes in demand, or Akamai CDN [11], which applies a DNS-based adaptive request routing mechanism for load balancing and web caching, applied on their host infrastructure. The described solutions are solving the existing issues for content distribution very well, but they are suboptimal, due to high hardware costs and the lack of change-dynamics in the server infrastructure, which has to occur very fast. Another issue relates to the network distance between server and client, the so called communication latency. Even if the server is able to handle all client requests alone, the big network distance from server to client can cause big communication delay. Using servers with such communication delay (e.g. users in Switzerland, which are connecting to a server in Sweden, can be connected through a major peering point in the USA), can increase the load on important links (e.g. a transatlantic link), but can also result in increased transmission delay. Today's CDNs have significantly contributed to more flexible and efficient content distribution and content availability, but still there are many open issues regarding the problem of sudden network traffic spikes (flash crowds). The main problem of the insufficient adaptivity of current systems lies in the inherent unpredictability of fast traffic changes. Hardware and software constraints on the servers with respect to limited processing power and memory often cause poor performance. Usually, web servers are not capable of handling different hit ratios for different web content categories. The lack of performance in the communication protocols, like TCP/IP and HTTP can also be seen as a significant factor for poor web server performance.

1.4 Thesis Description

This Master thesis describes the design, the implementation and the evaluation of a novel CDN, codenamed Gezora [1]. Gezora is a server side application, which establishes an overlay network among the group of servers (usually volunteer-servers) and utilizes them for temporarily outsourcing the popular and frequently accessed web content. This is required in order to be able to fulfill the client's Quality of Service (QoS) requirements during the peak request rates, where the origin server is not able to handle such a request number at once. The users QoS requirements generally consist in guaranteed content delivery and a low request time. The volunteer servers, distributed on different geographic locations, also called surrogate servers are used not only for content outsourcing, but also as the contributors for the request routing and load balancing. More precisely, the volunteer servers in the Gezora CDN are able to change their role dynamically, at the run-time, dynamically shifting from the role of pure content provider, where

a server retrieves 100% of client requests, to the double server's role, where a server acts as content provider and a reverse proxy (load balancer) at the same time, and finally to the pure load balancer role, where a server redirects 100% of the client requests. Using this approach, Gezora provides a self-adaptive and on-demand scalable CDN. The state of the art real-time request rate monitoring system, which consists of an Exponential Wighted Moving Average (EWMA) filter, used for the request frequency calculation, and of the feedback control loop, used for weight adaptation, are responsible for these properties of Gezora. Gezora addresses the current CDNs issues, such as the issues in the current Internet transport protocols (e.g. TCP/IP), as also the hardware constraints (large number of CPU interrupts, slow I/O, low network bandwidth), that are mostly exhibited at high client request rates. The novel request forwarding mechanism implemented in Gezora is based on the Random Early Detection (RED) [3] algorithm, a commonly used algorithm for congestion control in the TCP/IP transport protocol [12]. But instead of controlling the packet congestion, we concentrate on client requests congestion. This algorithm is combined with the simple URL rewriting mechanism, responsible for changing of the URL address from the origin host name to surrogate and the HTTP redirection mechanisms, which was used to force the clients to temporary request the surrogate servers. All described mechanisms make Gezora CDN very robust to sudden traffic spikes.

1.5 Thesis Contribution

We designed and implemented the first Gezora prototype using J2EE technology and Java servlets. We found Java servlets to be the most appropriate technology for the Gezora prototype implementation, since the HTTP requests can be handled easily using Java servlets. This allowed us to implement the efficient frequency monitoring mechanisms used as basis for the request redirection logic. The second Gezora prototype was designed and implemented in the C/C++ programming language in the kernel space of the Windows Server operating system. For this implementation, we used a MS Windows server operating system and an Ethernet packet capturing library, called WinPcap [13]. The second Gezora prototype differs from the first one in different architecture, which focuses more on the raw Ethernet packet monitoring, compared to the Java prototype, where the HTTP request monitoring was used. In fact, the main idea behind the second approach is nearly equivalent to the first one, since the HTTP request frequency is also monitored, but here the monitoring occurs in kernel space, which is positioned in a different network layer (Layer 2), compared to the first approach, where HTTP request frequency monitoring occurs on the application layer (Layer 7). Due to the complexity of TCP/IP, which is the core communication protocol in HTTP, the second approach did not function as expected. Nevertheless, it was an interesting attempt to pursue load balancing on the network layer 2, which may be further studied in the future.

The preliminary evaluation of the RED based request forwarding in Gezora was accomplished using a virtual Java based client model, which was able to simulate client requests with a variable request rate per time unit as a parameter. The preliminary tests gave us the general feedback about the RED efficiency and approved the next steps in the Gezora prototype develop-

ment. The main evaluation of the Gezora prototype is accomplished using the web stress tests. With this evaluation method, we were able to measure various parameters during the client-server communication, such as the client request time duration in ms, the % of clients waiting for web content fetching during some time period, and much more. The principle of web server stress test is very simple. In such tests, a large number n of virtual clients with predetermined URL address and access frequency rate are generated. The clients are polling the content from server with variable rate and the system measures the time communication gap between server and client, in order to assess how long the client had to wait for accessing the content.

In the second evaluation scenario, we compared two systems: a system with one origin server and a system with two servers, connected with the Gezora server application. The evaluation results show, that the system with the Gezora had a 30% lower average request time than the the origin server alone, during the high load rate. The difference between system with and without Gezora shows also the limits of a single server when handling a big number of requests. A single server had large response times, when the number of requests was bigger than a 1000 requests/s. This shows Gezora's benefits regarding the on-demand scalability with a minimum number of deployed surrogates.

1.6 Thesis Outline

This thesis is organized in the following manner: Section 2 provides a detailed insight into the world of CDNs and other aspects in their development. It also identifies the unique qualities associated with CDNs, which distinguishes them from other distributed computing paradigms and presents the structure, classification of CDNs as also the insight into the current content distribution and management techniques. Section 3 describes similar academic CDNs, such as: CoDeen, Coral CDN, Shark and Jellyfish. Furthermore, this section describes the following technologies used in Gezora:

- Exponential Weighted Moving Average (EWMA)
- Feedback Control Loop
- Random Early Detection Algorithm (RED)
- Request Forwarding Mechanisms: HTTP Redirection / URL-Rewriting

This section gives also a brief introduction into the Java servlets technology framework (a vital part of J2EE), which is used for the implementation of the first Gezora prototype. In Section 4, we describe the conceptual design of Gezora. Section 5 describes the Gezora prototype implementation with Java Sevlets and J2EE as also the implementation of the second prototype, accomplished using Ethernet packet capturing framework, called WinPcap [13]. Section 6 categorizes the general CDNs performance evaluation methodologies, describes evaluation scenarios as well as the evaluation results. Finally, Section 7 outlines the conclusion and the possible directions in the future development process of Gezora.

Chapter 2

Structure and Classification of Content Distribution Networks

2.1 Introduction

Content Delivery Networks (CDNs) have been designed to offer improved performance with respect to bandwidth and accessibility through content replication. CDNs offer fast, reliable services by redirecting the web content to cache or edge servers situated close to the users, as also a number of features and benefits, including content-delivery, request forwarding and distribution, as well as complex accounting structures. CDNs usually consist of a set of edge servers (which can also be referred to as surrogates), that deliver copies of the original content to the end users. The request forwarding structure is responsible for client requests redirection to the appropriate edge servers. It exchanges the information with the distribution system to maintain the current record of the web content stored in the CDN's caches. The distribution system has the task to transfer web content from the origin server to the CDN's servers, while the accounting infrastructure looks after a detailed usage record of the various clients, which have had accessed the CDN servers. This information is used to create traffic usage reports. Typically, CDN's host static content such as images, media and video material such as advertisements, and other objects which can be embedded for dynamic web content. CDNs are regularly used by Internet and media companies to advertise their products, as well as data centers, Internet Service Providers (ISPs), on-line music retailers, mobile phone operators, manufacturers of consumer electronics, and hosts of other businesses who need to publish content on the Internet in a secure and timely manner. CDNs focus on providing the following services:

1. Network infrastructure for the storage and management of content
2. Distribution of content among the surrogates and content cache management
3. Timely delivery of the static, dynamic and other streamed content
4. Network workload monitoring, performance management and reporting

An example of CDN, where the web server surrogates are located on the various locations, is shown in Figure 2.1.

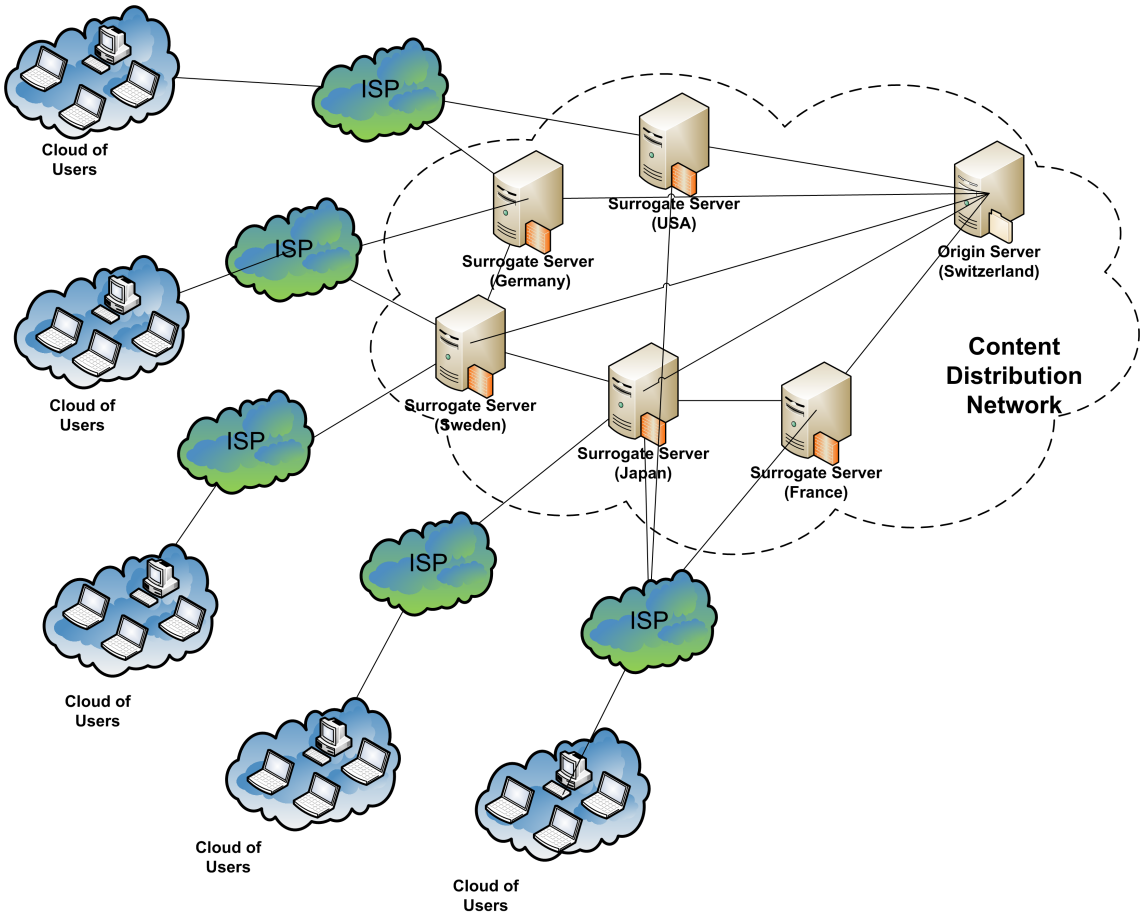

Figure 2.1: Content Distribution Network Example

2.2 Primary CDN Goals

Today's CDN providers guarantee the fast delivery of digital content. They host a variety of third-party content, including static content such as HTML pages, images, files, and software patches, as well as streaming media (e.g. audio and time video), and providing content services, like directory services, e-commerce, and file transfer services. These third parties include large enterprises, a wealth of web service providers, media companies and news broadcasters. The end users are able to communicate with the CDN in a number of different ways, by specifying the content or service request via mobile devices or a desktop computer. Naturally, CDN providers charge their clients in accordance with the content or traffic delivered to the end users

via their surrogate servers. CDNs support complex accounting mechanisms which collect and track information on the client's usage, relevant to the request forwarding, as well as the distribution and delivery. This system gathers information in real time for each CDN component. The information can be used within the CDNs for account billing and for maintenance purposes. The cost for charging of CDN services is high, often beyond the budget of most small to medium enterprises or not-profit organizations. The factor which most influences and affects the costs of CDN services include:

1. The bandwidth costs
2. The variety of traffic distribution
3. The size of the content replicated across the surrogate servers
4. The number of surrogate servers required
5. The reliability and stability of the system as a whole and the issues of security when outsourcing content delivery

CDNs are designed to enable content providers (or customers) to guarantee a high level of QoS to the end users who access their web content. As pointed in [7], CDNs focus on the optimization of the following CDN parameters:

1. Scalability
2. Content transfer reliability
3. Responsiveness and performance on peak request rates

2.2.1 Scalability

The primary CDN goal is to achieve scalability. Scalability is the term to describe the ability of the system to expand in order to handle large increases in data, users and transactions without any noticeable decline in performance. To be able to expand on a global scale, both time and money needs to be invested by CDNs in securing additional network connections and infrastructures. This provisioning of resources is required to dynamically address flash crowds and varying levels of traffic. A CDN has to be able to absorb the increased traffic levels, caused by flash crowds, by automatically providing capacity on demand. This increased capability allows a CDN to avoid costly over-provisioning of resources and guarantees a high level of performance to each user.

2.2.2 Content Transfer Reliability

Reliability can be defined as the service availability and the boundaries to which a service outage might be expected. A CDN provider can improve customer access to confidential content by delivering it from a multitude of different locations. For this purpose, a network with fault-tolerance and the appropriate load balancing has to be implemented.

2.2.3 Responsiveness

Responsiveness defines the reaction to possible outages and how quickly the normal state is reached in the course of operation. The responsiveness can also mean how fast CDNs can adapt to the changes (i.e. request frequency changes).

2.2.4 Performance

The performance of a CDN can generally be described by the response time of requests delivered to the end users. A slow response time is the single greatest contributory factor in customers abandoning web sites and their processes. The overall reliability and performance of a CDN is influenced by the location of the web content and request routing mechanism, as well as the replication of data and the caching strategies.

2.3 Requirements on CDNs

CDNs assume that in a network with a large number of nodes, there are some willing to cooperate, with their owners providing web resources voluntarily to the network community. From this assumption a conclusion can be drawn which illustrates how diverse these nodes are in terms of hardware, software and network characteristics. Most probably the nodes in such a network are connected by a Digital Subscriber Line (DSL) or cable provider, as end user machines are not expected to be highly available. Many nodes are in fact mobile devices, such as laptops or mobile phones, what in turn makes the nodes usually available for at most a few hours as shown by some researches. The goal of a CDN is to replicate web content in order to increase performance and availability, what raises the issues, assumptions and tasks listed below, which must be taken into the consideration when contemplating the design of the system:

- A CDN should appear to its clients as a single and reliable super server, enabling the clients to use standard web browsers with ordinary plug ins or applications, which run in the background, also called daemons.
- Each client should be automatically redirected to the replica server, that is the best for it considering client's characteristics and requests.
- If a lot of attention and importance is given to the document replication which has the performance as a main task, those documents should be placed close to their clients in order to achieve the desired performance.
- Multiple replicas require some sort of consistency management in order to determine when, how and which replicas are to be kept consistent.
- A CDN server needs to know something about the network topology and organization in order to be able to effectively replicate its documents, send and receive requests. This would most probably require some sort of brokerage system to permit automated resource discovery and allocation.

- Resource usage and provisioning should be fair among the nodes in order for them to collaborate.
- Joining a collaborative CDN should be easy and simple. Installing and configuring the necessary software should not require specific technical knowledge. All that is necessary should be a simple web based registration process, so the users would be able to register their machines and resources, make changes and if desired, terminate their membership.
- Security among the members should be enforced viciously so that malicious users can not attack the system.

  All these concepts should enable simplicity of distribution and replication of web content for users desiring to trust their site to a CDN, making the hosting of a user's web site transparent and no different than hosting it on a single machine.

2.4 CDN Components

There are three main components within a CDN architecture, being:

1. The content provider
2. The CDN provider
3. The end users

2.4.1 Content Provider

A content provider or client is is the component that delegates the URI name space associated with the web objects, that are to be distributed. The content provider maintains these objects on the original server.

2.4.2 CDN Provider and End users

A CDN provider is an entity which provides the basic infrastructure and facilities to the content providers, in order to deliver the content in a secure and timely manner. End users or customers are the bodies who access content from the content provider's website. CDN providers use caching and/or replicating the web content on servers, which are located in various locations. CDN cache servers are also referred to as surrogate or edge servers. Throughout this description we will use both of these terms interchangeably. The surrogates within a CDN are also generally called web clusters. CDNs distribute content to the surrogates in a way, that allows all cache servers to share the same content and URL. The customer's requests are redirected to the nearest surrogate, and the surrogate server which has been selected then delivers the requested content to the end users. This system provides transparency to the end users. In addition to the above, surrogates also send accounting information to the CDN provider.

2.5 Content Distribution and Management

Content distribution and management are vital within Gezora for the efficient delivery of content and the overall performance. Content distribution includes the following occurrences:

1. The placement of surrogates at strategic positions: The main requirement is that edge servers are as close as possible to the various clients in terms of latency.
2. Content selection and delivery (based on the type and frequency of specific end user requests).
3. Content outsourcing (main requirement: timely content outsourcing/replication and minimum client waiting time).

Generally, content management is dependent on the techniques for cache organization (i.e. caching techniques, cache maintenance and cache update). In [7], content selection and replication mechanisms are summarized. We rely on this work to briefly describe various mechanisms for efficient content selection and delivery in CDNs.

2.5.1 Content selection and delivery in CDNs

The efficient delivery of content is determined by the right selection of content to be delivered to the various end users. A suitable approach to content selection can assist in reducing the client download time and the load on the server. Content can be delivered to the customers in full or in parts.

2.5.1.1 Full-site content selection and delivery

Full-site content selection is a very simple approach, where the complete set of ressources is outsourced to surrogate servers. In short, with this approach, the surrogate servers perform the entire replication to deliver the total content of the site to the end users. A content provider configures its DNS in such a way, that all requests are processed by a CDN server. The main advantage is the simplicity of this approach. However, such a solution is not feasible considering the ongoing increase in the size of many web objects. Furthermore, as the web content is not static, the problem of having to update a huge collection of web objects is unmanageable.

2.5.1.2 Partial site content selection and delivery

As an alternative, a partial-site content selection uses surrogate servers to perform the *partial replication* to deliver the embedded objects only. The embedded objects can be web page images, or other static content. With partial-site content delivery, a content provider modifies its content so that links to specific objects have host names in a domain

for which the CDN provider is an authority. Therefore, the base HTML page is retrieved from the origin server, while the embedded objects are retrieved from CDN cache servers. The partial-site approach is better than the full-site approach in respect to the fact that the former approach reduces the load on the origin server and on the site's content generation infrastructure. Furthermore, the infrequent change of embedded content means, that a partial-site approach exhibits better performance.

The selection of content is dependent on a suitable management strategy for the replication of web content. Assuming that the approach is to select embedded objects so as to perform replication, the partial-site approach can be further divided into:

1. Empirical-based Replication
2. Popularity-based Replication
3. Object-based Replication
4. Cluster-based Replication

2.5.1.3 Empirical-based Replication

In the empirical-based approach, the administrator chooses, which content is to be replicated. Heuristics are used in the making of such an empirical decision. The main drawback of this approach is the uncertainty in choosing the right heuristics.

2.5.1.4 Popularity-based Replication

With the popularity-based approach, the popular content is copied to the surrogates. This process needs more time, because the statistics about the specific access patterns needs to be generated periodically, as the popularity of each content category changes. In addition, statistics are not always available for the new content category.

2.5.1.5 Object-based Replication

In the object-based approach, the content is replicated in units of objects. This approach is greedy, as each object is replicated to the surrogate servers (under storage constraints). Despite the fact that a greedy approach can achieve the best performance, it suffers from high complexity when implemented on real applications.

2.5.1.6 Cluster-based Replication

In the cluster-based approach, web content is collected in groups based on either correlation or access frequency, and is replicated in units of content clusters. This procedure is performed either by specifying the cluster number, or by determining the maximum cluster amount. The content clustering can be divided into:

- **User's sessions-based content clustering**: With the user's session-based content clustering approach, the web log files collected from the server are used for clustering of end user's navigation sessions that show similar characteristics. This approach is highly beneficial in scenarios where the classification and clustering of the frequently accessed web content is highly required, as it helps to determine both the groups of users with similar browsing patterns and the groups of pages which have related content.
- **URL-based content clustering**: The URL-based content clustering [14, 15, 16] allows for the clustering of web content to be undertaken based on the structure of connections between pages on a website (e.g. hierarchical or full mesh structure). The popular content is determined from a web site, and is replicated in units of clusters where correlation between two URLs is determined using some correlation metric (e.g. correlation coefficients between URLs, described in [17]). This URL-based content clustering is a very important activity at web mining [18] and URL normalization [19] techniques. Evaluation results show that content replication based on the URL clustering approach reduces the client download time as well as the load on servers. However, these schemes suffer from the complexity involved in deploying them. For example, the web site can be developed in such a way that one URL address has dependencies to other URL addresses, which are not clustered at that moment.

2.5.2 Content outsourcing

For the content outsorcing is very important that the right surrogate servers within a CDN infrastructure are selected for content delivery. Content outsourcing is performed using either cooperative push-based, non-cooperative pull-based and cooperative pull-based approaches.

- Cooperative push-based content outsourcing
- Non-cooperative pull-based content outsourcing
- Cooperative pull-based content outsourcing

2.5.2.1 Cooperative Push-based Content Outsourcing

This approach is primarily based on the pre-delivery of content to the surrogates. The content is pushed to the surrogate servers from the origin, and the surrogate servers communicate together to reduce the costs of the replication. This scheme allows the CDN to maintain a mapping between the content and the surrogate servers, as each request is forwarded to the closest surrogate server, or is otherwise routed to the origin server. With this approach, the greedy algorithm can be used to decide whether the content replication to surrogate servers should be accomplished or not. However, this is still considered a theoretical approach, as it has not yet been adopted by a CDN provider.

2.5.2.2 Non-cooperative Pull-based Content Outsourcing

In the non-cooperative pull-based approach, the requests are forwarded to the nearest surrogate server, using either a DNS redirection or a URL rewriting . If a cache miss occurs, the surrogate servers extract content from the origin server. The most popular CDN providers, such as Akamai [11, 20] and Mirror Image [21], use this approach. The main drawback of the non-cooperative pull-based content outsourcing approach is that an optimal server, regarding the communication latency (network distance), is not always chosen to serve the content request. The cooperative push-based approach is still at an experimental stage.

2.5.2.3 Cooperative Pull-based Content Outsourcing

In the cooperative pull-based approach, client requests are directed to the nearest surrogate via DNS redirection. Surrogate servers are communicating with each other to obtain the requested content in the event of a cache miss. Using a distributed index, the surrogate servers locate nearby copies of requested content and store it in the cache. The cooperative pull-based approach is reactive, whereby a data object is cached only when the client requests it. An academic CDN, named Coral [22] has implemented the cooperative pull-based approach using a variation of a Distribution Hash Table (DHT).

2.6 Request Forwarding in CDNs

A request forwarding system is responsible for routing requests to a suited surrogate server for the delivery of content. It directs requests to the replica server nearest to the client, although the nearest server may not be the optimal surrogate server for the servicing of the request. A request forwarding system uses various metrics, such as the client perceived latency, network proximity and replica server load in order to direct end users to the nearest surrogate that can best serve their request. The content selection and delivery techniques, such as full-site and partial-site, used by a CDN, have a direct impact on the design of the request forwarding system. Alternatively, if a partial-site approach is employed, the request forwarding system has to be designed so as to receive the customer's requests, so that the origin server can deliver the basic content while the surrogate servers deliver embedded objects. The request forwarding system in a CDN has two constituent parts:

1. Request forwarding algorithm
2. Request forwarding mechanism

A request forwarding algorithm is activated upon receipt of a customer's request. This specifies how an edge server should be selected in response to the associated request. A request forwarding mechanism is a method, by which the client is informed of the selection. The mechanism initially invokes a request forwarding algorithm and then informs

the client about the election result it obtains. A typical request routing mechanism in CDN is shown in Figure 2.2.

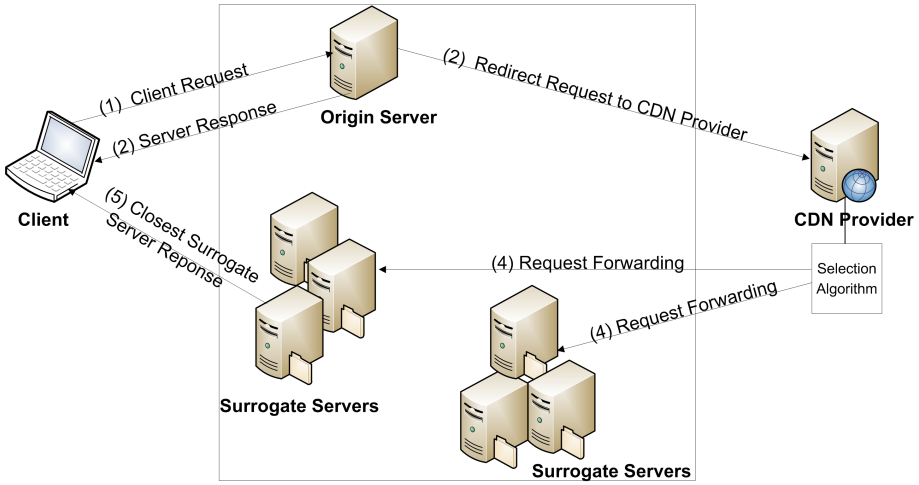

Figure 2.2: Request Forwarding in Content Distribution Networks

Figure 2.2 illustrates the following interaction flow in a CDN during the server-client communication:

1. The client sends one HTTP request using the URL-address. Initially, this request is sent to its origin server.
2. When the origin server receives this request, it makes a decision to provide only the basic content, such as the index page of the web site, which then can be served from its origin server.
3. To serve high bandwidth demanding and frequently accessed content, like pictures, the origin server, which initially distributes the content, redirects the client's request to the relevant CDN provider.
4. By using the proprietary selection algorithm, the CDN provider then selects the replica server nearest to the client to serve the requested embedded objects.
5. The selected replica server receives the embedded objects from the origin server, and then serves the client requests and caches it for subsequent request servicing.

2.6.1 Request Forwarding Algorithms

The algorithms activated by request forwarding mechanisms can be categorized as follows:

- Adaptive request forwarding algorithm
- Non-adaptive request forwarding algorithm

2.6.1.1 Adaptive request forwarding algorithm

The adaptive request forwarding algorithm monitors the current system state, and accordingly decides whether to select a cache server for content delivery or not. The current system state is obtained by estimating a number of metrics, such as the current load on surrogate servers, or the traffic congestion. The complexity of adaptive algorithms comes from their ability to change behavior in order to cope with an enduring situation. An adaptive algorithm demonstrates high system robustness in the face of events like flash crowds. Akamai [11] and Jellyfish (c.f. Section 3.4) CDNs apply adaptive request forwarding algorithms.

2.6.1.2 Non-adaptive request forwarding algorithm

The non-adaptive request forwarding algorithm uses heuristics for the selection of a cache server rather than considering the current system condition. A non-adaptive algorithm is very easy to implement, while the former is significantly more complex. Non-adaptive algorithms work more efficiently when the assumptions made by the heuristics are met. A commonly used non-adaptive request forwarding algorithm is the Round Robin algorithm [23]. To achieve optimal load balancing, a non-adaptive request forwarding algorithm considers all surrogate servers in the CDN, giving every surrogate server a certain time slot, where the corresponding resources can be used. The non-adaptive request routing mechanism has also some disadvantages. The first disadvantage is, that this approach does not consider the latency (network distance) from clients to surrogates. The second disadvantage is, that the non-adaptive request forwarding is not able to distinguish different types of client requests.

2.7 Similar Systems to CDNs

Four distribution systems have similar characteristics with CDNs. These are:

- Data grids
- Distributed databases (DDB's)
- Peer-to-peer Networks
- Multicast Networks
- Content Centric Networks (CCN's)

2.7.1 Data grids

A data grid is a data-intensive computing environment that provides services to end users in different locations so as to identify, transfer and manipulate large repositories of stored data. In the minimum application, a data grid provides two basic functions:

high-performance, secure data transfer, and scalable replica discovery and management. A data grid is composed of computational and storage resources placed in different locations, which are connected by various high-speed networks. Data grids are particularly orientated to large-scale scientific applications, such as high energy physics experiments at the Large Hadron Collidor [24], astronomy projects - Virtual Observatories [25], and protein simulation, e.g. BioGrid [26], which require a huge amount of data analysis. The data generated from an instrument, an experiment or a network of sensors is subsequently stored within a principle site and transferred to other storage sites around the world on request through the data replication mechanism. End users access the local replica records for locating of the data which they need. Once the appropriate permission and rights to access have been granted, the dataset is collected from the local repository, if present, or it is fetched from a remote repository. The data may be transmitted to another computational unit for processing. After the data processing is accomplished, the outcome is sent to a shared repository or to individual end users desktops. The resources found within a data grid are heterogeneous and can be distributed over several domains. The large datasets, distributed data collections sharing, which have the same logical name space, and the restricted distribution of data can be considered as the unique characteristics of data grids. They also contain some application specific characteristics. The overall goal of data grids is to bring together existing resources in order to achieve a certain performance benefit through data distribution. Data grids are invariably created by institutions which come together to share resources with common goal, by forming a Virtual Organization (VO). An example of a data grid is shown in the Figure 2.3.

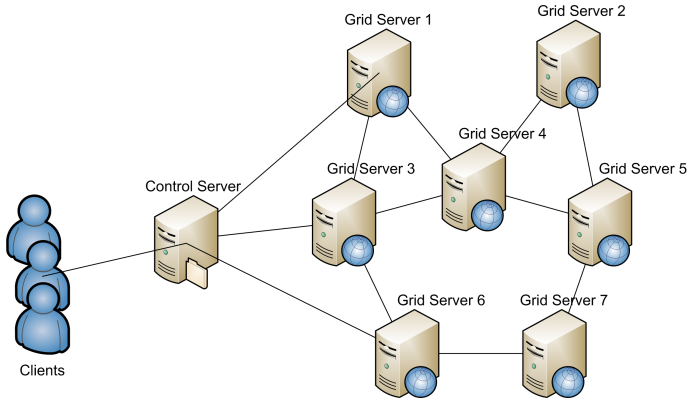

Figure 2.3: A Data Grid Architecture

2.7.2 Distributed databases

A distributed database (DDB) is a collection of logically organized databases and their corresponding data, distributed across a number of physical locations. A distributed database may be stored within several servers located at the same physical location, or dispersed across a wide network of interconnected computers. Each of the computers in a distributed database system acts as a node. In a distributed database, a node acts as either a client, a server or both, depending on the situation. Each location has a reasonable degree of autonomy and is capable of processing local queries, as well as participating in the execution of global queries. A distributed database can be constructed by separation of a single database or by unification of existing databases. The distribution of a system in this manner is transparent to the end users, as they interact with the system as an isolated logical system. DDBs have primary evolved to serve the needs of large organizations as a more performant solution than a single database systems. The main idea was to interconnect the existing databases and add new databases to organizational units. The usage provided by DDB includes the transaction distribution process, query and optimization, and the efficient management of resources. DDBs are dedicated to integrate existing and diverse databases and to provide a uniform interface for query processing with increased security and throughput. One example of distributed databases are the Oracle Exadata Storage Servers [27]. DDBs are similar with CDNs regarding the distributed transactions. But CDNs differ from DDBs, because CDN servers lacks autonomy, which is an important property in DDB web sites. Moreover, the main goal of CDNs is content caching, while DDBs are used primarily for query processing and transactions. The DDB architecture is shown in the Figure 2.4.

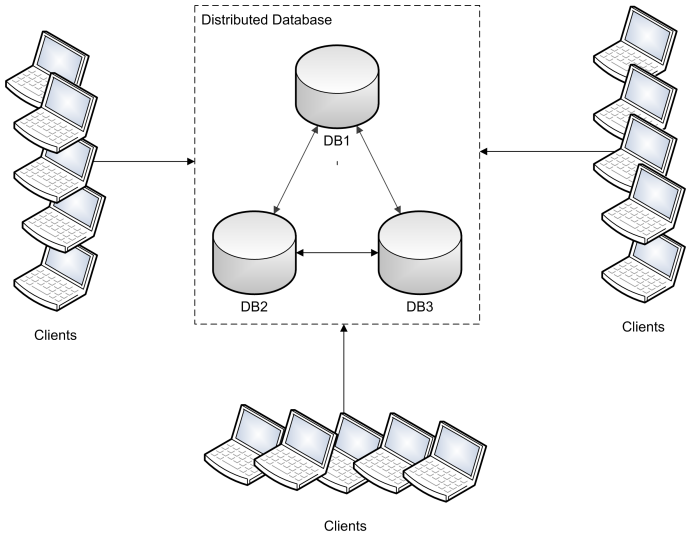

Figure 2.4: Distributed Database Architecture

2.7.3 P2P Networks

P2P networks [28] are primarily designed for directly sharing computer resources, as opposed to having an intermediate or central authority. Within a P2P system, each peer is autonomous and relies on other peers for resources, information and request forwarding. In an ideal environment, there would be no point of central control in a P2P network: Therefore, each of the participating nodes would collaborate to perform such tasks as searching for other nodes, the locating or caching of content, request forwarding, encrypting/decrypting, retrieving, and verifying the content. Today's P2P systems are also more tolerant on connection failures and scale better than the traditional centralized systems, because they have no single point which can be subject to failure. A single entity in a P2P network may join or leave at anytime. P2P networks are invariably suited to the individual content providers who are unable to access or to afford common CDNs. The content and file sharing found within P2P networks is mainly focused on the creation of efficient strategies so as to locate files within a particular group of peers, and to provide secure transfers of these files in case of high volatility, as well as managing high volumes of traffic (i.e. flash crowds), which are caused by heavy demand for popular files, in contrast to CDNs, whose primary goal is respecting the client's performance requirements, rather than the efficient searching of a peer with the required content and low network latency. Moreover, CDNs differ from P2P networks because the number of nodes which can join and leave the network per unit time is negligible, whereas the bandwidth rate is important in P2P networks. The known issues of P2P networks are: scalability problems arising at large numbers of participating peers, complexity (end user implementation required), security (peers cannot trust themselves inherently, few integrity checks), performance (due to the asymmetric network connections), routing overhead, and suboptimal use of available bandwidth (most P2P networks are not topology aware). The content availability is also questionable, since there is no guarantee that every content part is available. Figure 2.5 illustrates a classical P2P network architecture.

2.7.3.1 BitTorrent P2P Network

An example of such a system is BitTorrent [29], a P2P content distribution application, which has increased greatly in popularity in the last years. BitTorrent has certain advantages when used with peers, which have fast network connections (high upload rate) and it does have solutions for certain P2P issues, such as:

- Integrity checking
- Decentralization problem (uses Tracker and .torrent files for meta-data)

However, BitTorrent still has a problems with large, heterogeneous P2P networks (high upload utilization, fairness, etc.), which arises at distributing a big amount of smaller objects. The segmentation of many objects increases the routing overhead and does not scale well during the flash crowds, since the peers are interconnected with asymmetric

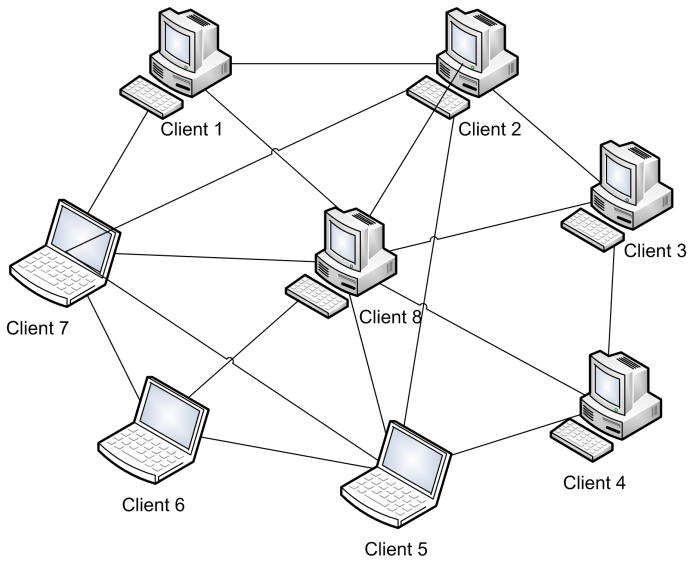

Figure 2.5: P2P Network Architecture

bandwidth connections, and they have different network latencies to each other. For this reason, BitTorrent has an advantage only when distributing large objects, such as ISO images.

2.7.4 Multicast Networks

Another content distribution approach, application level or IP multicast, can have benefits in the situations where a big number of clients fetches the same content at the exactly same time (e.g. video broadcasting of a football match). The successful deployment of the multicast paradigm requires to provide the same level of reliability and manageability as existing unicast networks, which is not the case. The complex installation, configuration and debugging is still one big issue. Another issue is the establishment of multicast groups for a specific content category. In case many small objects, multicast networks would not be able to hold their scalability, since the multicast trees would not hold the fast and reliable content replication to many clients at the same time, using only unicast connections in the background. Still, multicast networks are very interesting and may have many useful deployments in the future.

2.7.5 Content Centric Networks

Content Centric Networks (CCNs) are a quite new content distribution approach proposed by the American sociologist and philosopher, Ted Nelson and promoted by Van Jacobson, the research scientist from the XEROX Palo Alto Research Center [30]. The basic

idea behind this novel communication paradigm is that the data is observed as a primitive, whereas the content location is separated from its locality, identity, security and access. The communication is not host-name based (IP, net, subnet, host-name) like in today's Internet communication. It is based on the content's name. This changes the communication scheme significantly and moves the whole network stack from IP to a chunk-model, where the content is classified by name and accordingly separated in chunks. The communication in CCNs is regulated by content consumers. Here we distinguish two main packet types, namely the interest packets and data packets. The consumer, which is interested in one content type, is starting to broadcast the interest packets through the network. Any node that received interest packets, which contains the required content, responds with a data packet. Since at this communication paradigm, the content is exchanged by name, we call it *Content-aware* content distribution. The Figure 2.6 illustrates the CCN architecture and the work-flow during the interest request. The main advantage of this approach is that multiple nodes, which have interest in the same content, can share transmissions over the broadcast communication. Other advantages lie in the interest broadcasting for the content, which does not exist yet. This can activate a dynamic on-the-fly content generation in response. This makes CCNs ideal for unreliable packet delivery services as for mobile or ubiquitous computing. CCNs can also be applied in on-demand multimedia data distribution, such as video on-demand, Voice-over-IP. The main CCN disadvantage is the way of data retransmission. Unlike TCP, the CCN data-transport moves the responsibility to the content consumers regarding the data-retransmission. The content-consumers are responsible for re-expressing the interest for the content and it's retransmission, if they still require it. The content providers are further assumed to be stateless.

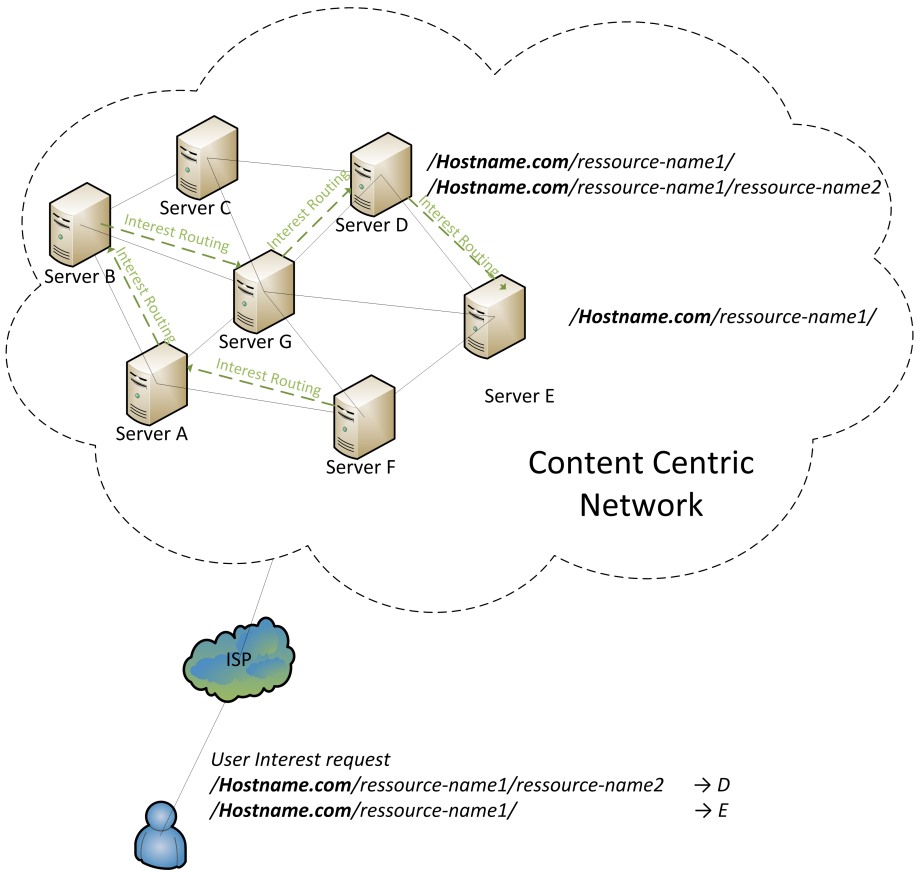

Figure 2.6: Content Centric Network Architecture

Chapter 3

Related Academic CDNs and Technologies used for Gezora CDN

3.1 CoDeeN

3.1.1 Introduction

CoDeeN [31] is a proxy server developed at the Princeton university, which is installed on the PlanetLab testbed [32, 33]. CoDeeN utilizes a group of caching web proxy servers in order to distribute and cache requests from a variety of clients. The system is open for a very wide array of clients, allowing the access from any client in the world. It is the most used service one the PlanetLab and it handles more than four million accesses daily. Usually, most open proxies have a number of problems: failing nodes, nodes performing malicious activities, node health, and much more.

3.1.2 Deployment of CoDeeN on PlanetLab and its Advantages

The Internet and wide area network testbeds have emerged in the past couple of years. The best example for this is the PlanetLab, which is mentioned in the introduction. The testbeds, like Planetlab have real traffic conditions. This is the most important requirement for the researchers. A CoDeeN is an attempt to utilize a decentralized design to address a latency sensitive problem. In order to address the latency reduction problem, other systems utilize geographically distributed server surrogates. These surrogates cache content from the origin server and request redirectors to send client requests to other surrogates. The advantage of CoDeen is that it is not a commercial testbed and as such it engages clients and not content providers. Because of this characteristic, CoDeeN is able to capture more information on the client's access behavior. CoDeen's future development might be going in the direction of the non commercial content providers which would be able to automatically send their HTTP traffic by transparent proxying, because of the infrastructural overlap. The CoDeen system is intended to consist of proxies operating in

both forward (intermediate surrogate accomplishes the content requests from the origin server, and retrieves them to clients) and reverse directions (surrogate servers act as load balancers or firewalls and are fully transparent to clients).

3.1.3 Architecture and Design of CoDeeN

CoDeeN is a latency sensitive environment, and the preferences are to avoid the problematic nodes even if they can give correct results. The reliability of CoDeeN CDN depends on the resource utilization during the run time. It is regulated by a timely state information exchange between various redirectors. According to the exact state of other proxies, the redirectors can decide which reverse proxies are used for the request forwarding.
The CoDeeN communication scenario can be described as follows: When a CoDeeN node accepts a client's request, it tries to satisfy it locally. In case of a cache miss, requests are handled by the redirector, which is determining where to send the request. Redirectors are usually other CoDeeN nodes which are utilized as reverse proxy servers (load balancers). CoDeeN is designed to use a node health monitoring mechanism for providing a reliable content distribution service. Using heartbeat messages, CoDeeN is able to estimate when the nodes can be utilized.

3.1.3.1 CoDeeN's Local Monitoring

In CoDeeN, local monitoring is used for collecting information about the state of CoDeeN's host environment. Local monitoring periodically checks the CoDeeN's primary resources, like CPU cycles, free file descriptors or queued incoming requests, but also the non critical information, like system load averages, classified traffic rates or free disk space. If a node does not have enough resources, it is marked as a failure node. To decide if a node is a failure node is not easy. There are many external factors involved in disrupting service health and they are usually outside of PlanetLab. Typically, the load averages of nodes are read every 30 seconds and updated inside CoDeeN, and the processor time in the operating system is also queried every 30 seconds, for up to 3 minutes. Some applications spend a lot of time inside the operating system but rarely more than 90%. All applications spending more time than 95% are considered as failures. Some faulty occurrences at nodes were also determined by the experience. For example, assume that some experiment consumed all available sockets. The local node was not only incapable to determine the existence of contact requests, but it was also unable to report the failure as well, since incoming requests were indefinitely queued inside the kernel. Home users can benefit from CoDeeN's local monitoring mechanisms on PlanetLab. Most PlanetLab nodes host some active projects at any time. Although this concept may lack virtual machines, it is still performing better than multitasking home systems.

3.1.3.2 Peer monitoring

In CoDeeN, peer monitoring is performed in two ways:

- By lightweight UDP heartbeats
- Heavy HTTP/TCP level helper

3.1.3.3 UDP heartbeats

UDP heartbeats are simple estimations to measure the status of the unhealthy peers. UDP is used when sockets are exhausted and as a result, TCP-based communication is blocked. Each proxy sends a UDP heartbeat message to the peer, which returns information about its local state. The generated UDP heartbeat traffic is small but in case of a need to reduce the traffic, the heartbeat frequency can be reduced. Each instance of CoDeeN independently assesses the health of other nodes by monitoring acknowledgments. If the acknowledgments are sent and arrive late, the node is determined to be a failure node. If the acknowledgments are sent and not received at all, the node is considered to be a dead node. This information is used to determine which node is a good candidate for handling forward requests.

3.1.3.4 HTTP / TCP Heartbeat

HTTP / TCP Heartbeat is a very useful method to determine the node health. Employing the tool to fetch pages over HTTP / TCP using proxy specifies what fails when the page can not be retrieved. The reasons are manifold and can range from socket allocation failure, slow / failed DNS lookup, incomplete connection setup and failure to receive data from the remote system. Each CoDeeN instance uses a dummy web server, as a good server of origin, so that the fetch tool could be able to test the proxying capabilities of the peers. The local node picks one of its peers to act as an origin server, tests all other peers as proxies using the fetch tool. During run time, all CoDeeN nodes will be tested once, and will have a function as origin server, used for proxy testing.

3.1.4 Differences To Gezora

The main difference between CoDeen and Gezora is that CoDeeN redirects the request only on a cache miss. Gezora accomplishes the request forwarding using the Random Early Redirection mechanism (described in Section 4.4), even if there is a cache hit. The amount of forwarded requests depends directly on the request frequency. At a very high request frequency, all incoming requests are redirected (100% reverse proxy modus). This makes Gezora more robust to fast changes in the client request rate than CoDeen. The request routing mechanism is accomplished in the same way in both systems, using a simple URL-rewriting mechanism.

3.1.5 Conclusion

CoDeeN does not have its own infrastructure like other commercial CDNs (e.g. Akamai), so it has to compete for resources with other experiments on PlanetLab sites. Depending on the general system usage, hardware resources might be exhausted as well as the network bandwidth, thus creating problems in availability, functioning and reliability of services provided by CoDeeN. The success of monitoring in the distributed environment depends on the difference between monitoring frequency and service failures. The majority of the failures in CoDeeN are longer than the monitoring frequency. Short failures can be avoided by the constant update of the node's history. Simultaneous requests are not used, because a forward proxy cannot determine whether or not the request has been sent to the origin server. Using the multiple parallel requests and waiting for the fastest answer is not an optimal solution, because it is often bound with errors. Unfortunately, practice has shown that the reliability of the CoDeeN system has decreased with more nodes being added. CoDeeN is currently operating on approximately 100 nodes and their status is much more dynamic and unpredictable than expected. Beside the technical disadvantages, CoDeeN has also an organizational problem. This problem arises in making the decision, which nodes should be allowed to act as open proxies. The nodes have not just to deal with the local requests, but with any client in the world as well, which led to the increased traffic. These issues emerging from open architecture include: system abuse, system unavailability because of the node disconnection, spam, service- and identity theft.

3.2 Coral Content Distribution Network

Coral CDN [22] is a decentralized and self-organizing CDN, similar to P2P networks. It allows clients to run a web site that offers high performance whilst meeting a huge demand. The costs for this are in the range of a medium-priced broadband Internet service. Coral CDN uses a latency optimized hierarchical indexing infrastructure, based on a novel abstraction called a distributed sloppy hash table - DSHT [34]. The positive side of Coral CDN is that its simple interface has led it to widespread application so that many sites began to use Coral as elastic load balancing solution, which dynamically redirects the network traffic. Coral CDN has been operational on PlanetLab since March 2004 and it was running until the end of 2009.

3.2.1 Corals Architecture and Design

Coral is built on top of the novel key-value indexing infrastructure and is made of three stand alone applications:

1. Indexing Layer
2. Corals HTTP Proxy Servers
3. DNS Server

Web content is automatically replicated by volunteer sites as a side effect of users using it. In order to use a Coral CDN properly, clients just need to make a change to the visited web object's host-name in its URL and they will be redirected transparently to nearby nodes by a peer to peer DNS layer. The participating volunteer nodes work together in order to reduce traffic load on the origin server.

Two main characteristics make Coral ideal for efficient content distribution:

1. Coral allows nodes to locate nearby cache copies of web objects instead of querying for ones further away and secondly to prevent server overloading in the infrastructure.
2. Coral uses routing techniques popularized by P2P DHTs, where a traditional URL to node mapping is achieved. Still, it is a different approach than with DHTs, since Coral uses a novel Distributed Sloppy Hash Table, which lets nodes locate and fetch the required content from each other by name (similar to CCNs). The main idea behind this approach is that a node does not need to know the exact location of every replicated resource, it only needs a single, valid, nearby resource copy.

Reasons for using P2P DHTs in Coral are various. The most important are: locality, server overloading prevention, which is not possible on normal DHTs, and the fact that Coral's architecture is based on the clusters of well connected machines. Because of all these properties, it provides a weaker consistency than traditional DHTs. For this reason, Corals DHT can be called Distributed Sloppy Hash Table (DSHT). Unmodified web browsers can operate with the system normally because Coral employs a P2P DNS redirection infrastructure and utilizes an algorithm for epidemic clustering, which exploits the distributed network. Epidemic clustering is a progressive exponential clustering, which correlates directly to the number of established connections between peers. This phenomenon is very similar to the spreading process of the human virus, which also correlates to the number of human contacts.

Coral CDN consists of three main parts:

1. Network of DNS name-servers for *nyud.net* with the main task of mapping clients that are near to Coral HTTP proxies.
2. Underlying Coral indexing infrastructure and clustering machinery, with the main task to represent the platform for other two components.
3. A network of cooperative HTTP proxies that handles requests from users.

Coral can be used by publishers, third parties or end users, without any software installation. The system functioning can be outlined in the next steps:

1. When the client sends the DNS request to the Coral name-server, Coral probes the client to measure the round trip time and uses this information to determine an appropriate name-server and proxy to return.
2. The client sends the HTTP request to one of the returned proxies and if a web object is cached locally, the process is finished. If the web object is not cached locally, the

proxy attempts to find the object on another Coral CDN proxy, which is nearest to the client.

3. The proxy searches for the object's URL in the Coral indexing layer and if the object is found, the proxy returns it to the client from the node where it is cached. If the object is not found than the proxy fetches it from the origin server.

4. After the process, the proxy stores the reference of it self and the object in Coral, announcing the fact that it is now caching the object.

3.2.2 Differences To Gezora

The main difference between Coral CDN and Gezora is, that Coral requires user action for for changing of the URL address, which points to the Coral CDN nodes. The redirection is accomplished using a P2P DNS layer. On the other side, Gezora uses a simple URL rewriting and HTTP Redirection for Request routing and does not require any action in order to use it. It is fully transparent to clients.

3.2.3 Conclusion

Coral CDN being a P2P web content distribution network utilizes people's willingness to redistribute data, which they find useful. Indexing of cached web content is achieved by distributed storage abstraction called a DSHT. Traffic congestions are avoided by DSHTs mapping of a key to multiple values and scaling many stores of the same key. Node clustering by network diameter in Coral CDN can be described as follows: Each Coral node is mapped to several distinct DSHTs called clusters. Each cluster is denoted by a maximum desired network round trip time (RTT), also called a network diameter. Furthermore, it uses a hierarchical structure of fixed diameters. At each hierarchy level, every node is a member of one DSHT. A group of nodes can actually form one cluster at one hierarchy level (level a), if their network diameters are below the diameter threshold at the same hierarchy level (level a). Coral uses a three-level hierarchal structure. Some similar approaches are described in [35]. Successful clustering of nodes by network diameter ensures that nearby data replication can be located and retrieved by Coral, without querying more distant nodes. A peer to peer DNS layer redirects clients to nearby Coral proxies and allows unmodified web browsers to benefit from Coral CDN and in turn avoid overloading origin servers. Disadvantages in the Coral CDN architecture is an intention to trade the server load for latency and in the process mask the temporary failures at origin servers. Coral CDN may not be ideally suited for smaller scale platform like PlanetLab [36, 37], but it provides the interesting self-organizing and hierarchical properties.

3.3 Shark

3.3.1 Introduction

For accessing remote data, network file systems can offer a transparent and very powerful interface to a wide public. In the current network file systems like NFS [38], clients fetch data from a central server, which is connected to other servers in the network file system. This, unfortunately, limits the system's ability to provide the service for a large number of clients, since the single server cannot handle all requests at once during the high load rate.

Recently developed systems, designed as peer to peer systems, have managed to reduce this scalability bottleneck, but they are often too complex and provide non standard models for accountability and administration. The designers of the Shark [39] system have tried to give to the wider public the scalability of distributed systems with the simplicity of systems with central servers. The Shark system is an attempt to merge the best characteristics of a distributed file system for large scale - wide area deployment, with characteristics of local area file systems.

Shark introduces a new concept, where clients mutually exploit each others file caches in order to reduce the load on the origin server. Clients can find nearby copies of data using a distributed index even if the files are located on different origin severs. Shark has proven that it can retain competitiveness for clients in local area network while reducing the latency for reading heavy workloads in both local and wide area networks. To summarize, it can be stated that Shark has the ability to help servers with modest characteristics, service hundreds of clients while keeping the usability, consistency and accountability, on a satisfactory level.

3.3.2 Use Cases

Users of distributed environments often launch similar processes on hundreds of machines almost simultaneously, what can create difficulties in debugging. Distributed web hosting and network measurement require software with low level network control and resource allocation. Testbeds such as PlanetLab [36, 37, 32, 33], provide users a possibility to replicate their programs along with some minimal execution environment before they launch a distributed application.

Shark is especially designed to provide a support for widely distributed applications. Rather than to manually replicate program files, users can place a distributed application and its entire run time environment in an exported file system and execute the program directly from the file system on all nodes. Shark deals successfully with a large number of client requests by locality aware cooperative cache. Shark will enable clients to download nearby cached copies of identical desired files or even chunks of desired files from other clients instead of contacting the origin. Shark coordinates client caching by leveraging a locality aware, P2P distributed index thus forming the client machines into self organizing clusters.

Clients, who attempt to simultaneously read identical data, are directed to nearby replicas and can fetch the content from each other in parallel. This represents the main idea of the Shark system. Because of its characteristics, Shark has become a satisfactory tool in environments where other systems have been previously inefficient. Shark and local hosts interact by using existing network file system protocols, but Shark runs in user space.

3.3.3 Design

The design of the Shark system incorporates several ideas aimed at reducing the load on servers, thus improving client perceived latencies, enabling clients of mounting remote file systems and efficiently accessing them. The Shark system enables the client's equipment to become multi-functional machines. They act as a client (when handling a local application file system access), as a proxy (when they are serving cached data to others), or as a node (within the distributed index overlay). Generally, such a system would traditionally use different machines for every role.

When the client fetches the file from the origin or a nearby proxy, Shark caches it and registers itself in the distributed index. When another client attempts to access the content, it discovers the pieces of the content near itself and downloads the content in parallel via a secure channel and then registers itself in the distributed index, thus becoming another available source and decreasing the load on the origin server. A Shark server divides a content into pieces by using the Rabin fingerprint algorithm [40].

A client discovers replica proxies with the content pieces via Shark's distributed index and other Shark clients, which cache content pieces, organize themselves into a key / value indexing infrastructure (built on a peer to peer routing overlay) in order to enable a client to retrieve the desired content. All security issues that might compromise the process (like a malicious proxy breaking data integrity and altering the content) are regulated by tokens generated by the file server.

If a malicious proxy compromises the data, the client is able to detect a change. Tokens serve as a shared secret between the proxy and the client for deriving keys for data transmission. Indexing keys are only derived from a token and they do not compromise in any way a token's shared secret. Shark allows the clients to freely exchange data segments that are not entire files without use of the tokens. In this way, one can still act as a proxy and transmit data to others, because small portions of the files are identical even if the file versions are different.

3.3.4 Differences to Gezora

The Shark system is very similar to P2P Networks. The most important feature is that the client, which fetches some content, caches its parts locally. The content chunks are being discovered from the other clients, if they fetch the same web content, and are positioned very near (i.e. latency) to the client with the same content chunk. On the other side, Gezora does not use client machines for content caching. The content is being replicated

only on volunteer servers, which are the part of the Gezora's cloud infrastructure, similar to CCNs 2.7.5.

3.3.5 Conclusion

The designers of the Shark system have attempted do create a network utility file system, which can provide services to hundreds of clients, while simultaneously replacing local area file systems. Shark exploits existing local file systems, ensures compatibility with existing administrative procedures and provides a performance, which is competitive with other secure networks beyond expectations.

Clients on a Shark system form a locally optimized cooperative cache by constructing self-organizing clusters of very well connected machines in order to achieve improved wide area performance. They efficiently locate nearby copies of data using a distributed index. They download files or parts of the files from multiple proxies in parallel. This reduces the load on the origin servers and delivers much improved (much reduced) latency for the clients. Because of all these characteristics, Shark achieves a scalable, efficient, secure yet easily administrated file distribution system.

3.4 Jellyfish

3.4.1 Introduction

The expenses of running a popular web site are often in the range of hundreds of thousands of dollars. This is not affordable for many small companies, non-profit organizations and individuals. At the present moment there is no easy way for an Internet site to lighten the load of hosting to the volunteers, such as private PC owners. Practice has shown that many people would like to donate their idle computing resources in order to help a good cause, thus reducing the sites hosting costs. Ordinary PCs are geographically dispersed, unreliable, possibly malicious and could have low-bandwidth connection. One of the possible solutions is Jellyfish [41], a distributed web caching system, which is intended to run across ordinary unreliable PCs while maintaining a high performance. Jellyfish is the most similar CDN system to Gezora, since it allows volunteers to assist at request routing and content replication with available resources.

3.4.2 Architecture and Design of Jellyfish CDN

Jellyfish [41] is intended to be a cooperative, decentralized and P2P CDN. The scientists who designed Jellyfish have started from the assumption that systems for distributive caching are using the nodes, which are reliable in terms of malicious behavior, performance and bandwidth. This assumption will hold in the perfect environment where one overlay network has a big number of machines, which have a high bandwidth and processing power. The main requirement for Jellyfish was to retain the best possible content

distribution performance regardless of the degree of relative centralization. In most cases, decentralization implies the existence of the shared routing or object placement protocol, which provides the robustness and scalability at content delivery systems. Object's lookup and retrieval is the product of deterministic, non-hierarchical caching structure. On the contrary, a Jellyfish CDN is intended to use the hierarchical overlay caching structure. Generally, the utilization of the Jellyfish system is categorized by different groups:

- Content Providers
- Volunteers
- End users

3.4.2.1 Content Providers

In order to use the Jellyfish software, content providers (organizations or individuals) must supply at least one reliable and trusted machine on which Jellyfish can be run as a server. Jellyfish can be used only if the server software is properly setup to reconfigure their name servers without changing their actual code. A good side of the process is that one can create a sub-domain and make the changes on the main domain only when the sub-domain is fully tested and running.

3.4.2.2 Volunteers

Anyone who is willing to dedicate server resources, primarily broadband connection and to download the Jellyfish client can become a volunteer. Such persons select which website they would like to help and they can choose from a list of approved choices. When the machine of the volunteer is idle, it will connect to the main Jellyfish servers and register itself as an available node, what will in turn, immediately trigger the requests to be forwarded to it.

3.4.2.3 End users

It is intended that the users do not see any apparent difference between browsing the site conventionally and with the support of the Jellyfish. Unlike in Jellyfish, other systems (i.e. Coral) leave no choice for users to opt in or out of the caching system, because the content provider controls the choice. In the case of Jellyfish, content provider might make independently accessible cached site thus enabling users to choose which distribution system they would like to use.

3.4.3 Jellyfish Node/Super node Structure

The node/super node structure of Jellyfish is chosen to be based on a DHT overlay because the designers have felt that for the low latency applications this structural design

has advantages and achieves better performance. Some existing P2P applications like Skype [42], are applying the node/super-node structure. Generally, two key issues arise at distributed caching systems:

- A large portion of the PCs are behind restrictive firewalls, requiring the assistance of the unrestricted peers to initiate TCP connection with normal users.
- Big hardware and bandwidth differences at nodes in the overlay networks.

In Jellyfish, super nodes are considered to be all volunteer nodes, which are not behind firewalls and NATs with reasonably high capacities and uptime. Other volunteers without these characteristics are considered to be ordinary nodes. Super-nodes can act also as ordinary nodes or shift their role back and forth in time. Jellyfish assigns ordinary nodes to a super-node with a minimum network latency. The super-nodes can alternate their role as HTTP or DNS servers while the ordinary nodes can only act as HTTP servers without a choice.

3.4.4 Jellyfish Request Routing and Service Discovery

Request routing in Jellyfish is performed by using DNS, much like in many other systems. Generally, the entire cached web site is assigned to its own sub-domain (main HTML files, associated images, javascript and css files), which represent the simple translation of its URL. When the user sends the request for a page, the super-node determines which ordinary node should serve the request. If necessary, the super-node will also coordinate the communication between the user and the ordinary node, which acts as ordinary HTTP server. If the ordinary node has the requested content, it will answer the user. In another case, it will attempt to retrieve the content from a central server before forwarding it to the client. If the nearby nodes do not cache the content, a Jellyfish is designed to declare such requests as a cache misses, rather than searching for the cached content on the nodes, which are on the other side of the world. In order to find a solution to this problem and to reduce the number of cache misses, Jellyfish employs the cache digests, which are the bloom filter [43, 44, 45] based methods for efficiently reporting which content can be found in the cache of different nodes. Every super-node runs the bloom filter on its state representation of the cache content of its sub nodes, obtains the cache digest and sends the results to its nearby super-nodes. When a super-node receives the request that none of its ordinary nodes can fulfill, it starts to search among the cache digests of its super-node neighbors and forwards the client's DNS request to the super-node, whose ordinary node caches the file and creates the near cache miss in the process. The disadvantage of this procedure is that single bloom filters can give false feedback. The negative influence of this false feedback is acceptable, because it does not make much difference if it has not been used in the first place and the result would be the cache miss anyway. For eliminating the possibility of false cache digest results, some other methods can be employed, but those methods come with higher computational costs. After a described scenario occurs, the super-node, which originally got the request, will instruct his ordinary node to cache

the content, retrieved from the ordinary node of another super-node. As described, in this way the cache digest method is able to reduce costs and utilize resources more efficiently than gossip-based notification method deployed on some other systems.

3.4.5 Differences to Gezora

Jellyfish is the most similar CDN system to Gezora. It applies the very same principles of using the resources of the volunteer servers to establish a fully-distributed overlay network for efficient and load-insusceptible content distribution. Though, the hierarchical super-node/node structure as the bloom filter based cache digest differs from the principles of design proposed in the Gezora. At Jellyfish CDN, when the client sends a request for some web content, the super node is looking for the best appropriate node (in terms of load and network latency), which can deliver the content. The super-node can act either as the DNS- or a HTTP-server. This mechanism is very similar to Gezora's dynamically changing server roles (content provider / load balancer). Further, if the normal nodes do not cache the content, Jellyfish declares the client request as a cache miss. This method prohibits further searching for the content on the nodes, which have a bigger network latency (i.e. nodes on the other side of the continent). This can increase the response time, since the qualified cache miss triggers the further searching for the content and node-traversing. If the super-node receives the request that none of the other nodes can fulfill it, it search in the cache digest for it. This mechanism is very complex and can have benefits on large CDNs, but it's adaptivity to sudden traffic spikes is questionable. On the other side, Gezora does not use the hierarchical node structure, as also the cache digests, since it is not a pull-based CDN. Gezora uses a chained node structure, where every node forwards requests to other nodes, depending on the request frequency level. This system is much simpler and avoids the unnecessary node-traversing during one client request.

3.5 Java Servlets

3.5.1 Introduction

A Java servlet is a kind of server-side software, which handles the messaging between a client and the server. The Java servlet API [46] is used by to respond to the HTTP requests. The Java servlet API is an interface, used by a program to enable interaction with other software. It includes routines specifications, data structures, object classes and protocols, usually implemented by applications, libraries, and operating systems to determine their vocabularies, calling conventions and to enable access to their services.

The main characteristics of Java servlets are:

- Servlets are portable across platforms and across different web servers
- A servlet is persistent, because it remains in the memory between requests and it can maintain system resources, such as a database connection

- The servlet is loaded and created only once by a server, when the client initiates calls to a servlet, what makes it very efficient. Additional calls perform only the business logic processing unlike the CGI processes, which are loaded with each request, what decreases the performance. A Java virtual machine uses the lightweight Java threads to handle servlet requests unlike CGI, which uses the heavy operating system process
- A servlet is able to access a library of HTTP specific calls in order to benefit from the development of the Java language and servlet also separates the presentation from business logic. This separation makes it easier to split a project into pieces for easier maintenance and development

The Java class is a sort of blueprint or a template, used to create objects of the same class. This template describes the state and the behavior, that objects of the same class have in common. Speaking in more technical terms, the class is the cohesive package, which consists of a particular kind of meta data. Servlets may be uses to generate a dynamic content to a web server. Most commonly the generated content is a HTML, but it may be other data i.e. XML.

3.5.2 Servlet's Web Container

Servlets are representing the counterpart to non Java dynamic web content technologies. Servlets use HTTP cookies or URL rewriting in order to maintain the session variables during the server transactions. All API servlets, contained in the Java package hierarchy like javax.servlet, describes the expected interaction of a web container and a servlet.

Essentially, the web container is a web component, which communicates with servlets. In order to run a servlet the servlet container is needed. It is responsible for managing other aspects of the servlet life cycle such as:

- User sessions
- Class loading
- Servlet contexts
- Servlet configuration information
- Servlet persistence
- Temporary storage

Communication with the servlets occurs exclusively through a web browser, so servlet engines are useless on their own. They can be divided into three groups: those embedded into web servers servlet engines, which can be used as standalone web servers or connected to the other servers and as add-ons to the existing web servers. Some of the responsibilities of the web container are:

- Managing the life cycle of servlets

- Mapping a URL to a particular servlet
- Ensuring that the URL request has the correct access rights

3.5.3 Servlet's Communication Flow and Libraries

At servlets, the Java objects are defined by the servlet package, and represent the servlet requests and responses, as well as the objects, which reflect the servlet's configuration and execution environment. The javax.servlet.http package defines HTTP specific subclasses of generic elements, which include objects for session management. These objects include requests and responses between the server and client. Servlets have the ability to be packaged in a WAR file as a web application. A WAR file (web application archive) is a file used for distribution of: servlets, Java server pages, Java clases, XML files and static web pages, which all constitute web applications. Automatic generation of the servlets is possible with the Java server pages complier or by template engines such as WebMacro or Apache Velocity, in order to create HTML. Java server pages and servlets are used together in a pattern called Model 2. Model 2 is a kind of a model view controller pattern.

The following steps comprise the servlet life cycle:

- On start up, the web container loads the servlet class.
- The init method is called by the web container. This method initializes the servlet and must be called prior to the servlet serving any request. The init method is called only once within the whole life cycle of the servlet.
- Each request is served in its own separate thread. The service method of the servlet is called for every request by the web container. The service method determines of what kind the request is, and dispatches it to an appropriate method to handle it. It is the duty of the servlet developer to implement these methods. If a request is made for the unimplemented method, the parent class is called and the resulting error is returned to the client.
- After all is done, the web container calls the destroy method that takes the servlet out of the operation. Both init and destroy methods are called only once in the servlet's life cycle.

3.5.4 Building and functioning of Java HTTP servlets

In order to build the servlet, an editing environment is needed as well as the profound knowledge about object oriented programming and procedural programming. The compilation of the servlets requires a Java compiler and Java server development kit, which is an extension of the Java development kit. The Java servlet API [46] defines a standard interface for request and response messages. Because of this, the servlet can be portable across platforms and different application servers. Servlets have the ability to respond to client requests by dynamically constructing a response, which is sent back to the client.

Written in the Java programming language, servlets have full access to the variety of the Java APIs, what makes them ideal for implementation of complex business application logic and accessing data in different parts of the IT system's architecture. A servlet can be called multiple times to serve requests from many clients and can simultaneously handle and synchronize requests. Servlets can also forward the requests to the other servlets. A servlet runs inside of a special type of applciation web server to handle requests from other servlets, enterprise nodes and web applications. While both the web and application servers can run in the same machine, there is a big difference between them. The web server runs the client's code (applets) and the application server runs the servlet code. The server alone loads, executes and manages the servlets. Servlets are called either as an explicit URL reference or they are embedded in the HTML and called from the web application. It is not needed to execute a Java command to run a servlet, but instead the URL command is issued to point to the location of the servlet, which are generally located in any directory on a machine where an application server is running.

3.5.4.1 The Java servlet API

The Java servlet API [46] represents a set of the classes that defines the standard interface between a web client and a web servlet. Essentially the API encapsulates the HTTP requests as objects such that the server can pass them to the servlet and the responses are coated in the same way, so that they can be passed back to the client. The Java servlet API has two packages: the *javax.servlet* package, which contains classes to support a generic servlets and the *javax.servlet.http* package, which includes specific support for the HTTP protocol. The interface of the servlet class it the central abstraction of the Java servlet API. This class defines the methods that manage servlets and its client communication. A client request to the servlet is represented by the *HTTPservletRequest* object, which encapsulates the communication from the client to the server. It can contain the information about the client environment and data, which are sent from the client to the servlet. The response from the servlet to the client is represented by an HTTPservletResponse object, which is a dynamically generated object (HTML page). This response is built both with the data from the client's request and the data from the servlet's accessed resources. A simple example of the request/response scenario is a parameter passing by appending it to an URL. In this way, web administrators are able to track a session, what is extremely important when the client does not accept cookies. The servlet redirector takes the target URL and referring origin as an input and redirects the client's web browser to the target URL. Servlets override the doGet and doPost methods. HTTP GET requests are typically browser requests for web pages, which are issued when a user follows the link. The HTTP POST request is generated when a user submits an HTML form, which specifies the post and that in turn allows client to send data of unlimited length to the web server in a single attempt. The HTTP POST method is also useful when a client is posting information, which is confidential, such as a credit card number. The same servlet can handle both GET and POST methods, choosing between GET and POST calls. Other methods in common use are:

- Service - the lowest common denominator method for servlet implementation. However, developers mostly use the doGET or doPOST methods.
- doGET is used for HTTP GET requests.
- doPUT is used for HTTP PUT requests.
- doPOST is used for HTTP POST requests.
- doDELETE is used for HTTP DELETE requests.
- HTTP INIT and DESTROY are used to manage resources, which are held during the life of the servlet.
- getServletInfo is used by the servlet to provide information about itself.

Generally the two things that differentiate the free from the commercial servers are the complexity of the setup and configuration compatibility with the products and technical support. Typical examples of free web servers are: Apache Tomcat [47], JBoss Enterprise [48], etc. The today's commercial web servers are: MS Windows Server OS (IIS) [49], Red Hat Enterprise Linux Server OS [50] and others.

3.5.5 Security Considerations

Servlets do not have to handle security on their own, because they can rely on the capabilities of the web server to limit the user's access. The security capabilities of most web servers are based on the user and password or digital certificate for authentication, which determines the privileges. Web servers can use encryption in user name and password transmission via SSL connection or they can use more advanced methods (like digest authentication protocol, which transmits a hash of the user's password and server generated value). Both of these approaches are transparent to the end users and their browsers. Newer versions of the servlet API use the web.xml file, which can be used to define which servlets and resources are protected and which are allowed. This method divides users into categories and assigns roles to the users, which are the basis for their permissions.

3.6 Technologies and Algorithms applied in Gezora CDN

3.6.1 Introduction

Despite modeling errors and uncertainty, robustness and feedback control of QoS aware servers has gained much popularity. Server performance and availability is predominantly characterized by queues behavior. This behavior is a key element of the control loop. The main motivation for the understanding the behavior of queues in the context of feedback control schematics is their central role. The one part of the feedback control loop theory is implemented in Gezora, namely the Exponential Weighted Moving Average filter, described in the Figure 3.6.6, which is used for the request frequency noise filtering. The other parts of feedback control loop theory, like Weighted Fair Queuing, can also be a very

important component of Gezora. In [51, 52], the feedback control theory and weighted fair queuing are summarized. We rely on this work to briefly describe the impact of feedback control theory on request routing mechanisms.

3.6.2 Weighted Fair Queuing (WFQ)

The WFQ is a popular queuing method, when different traffic classes must be allowed access to the different share of a common resource on servers. The WFQ element brings many challenges, which make the simple feedback control ineffective, unstable and unreliable with a higher predictability and responsiveness. Robustness of simple feedback controllers with respect to modeling errors and large fluctuations in load on servers and resources has made control theoretic approaches very successful.

3.6.3 Impact of Feedback Control Theory on the Server Performance

The main reason for the control theory's ability for application across computing servers is that server queues can be modeled by difference equations and are amendable to control theoretic analysis. The server performance depends mainly on the flow of requests through the queues. Requests must pass through a series of stages first, where they are queued for service. Desired performance improvements can be achieved by understanding and changing the relationship between feedback controllers and queuing element. This is accomplished with the allocation of the server's bottleneck resource to each traffic class. A certain server performance is achieved by control of each portion of traffic class. The main disadvantage is that many resources (i.e. HDD head, CPU, system bus, communication channel etc.) can not be physically partitioned as disk space and memory. Feedback control theory determines the resource allocation to different classes by passing the appropriate weights to each class. Multi-class web traffic is believed to be more generally applicable and intended for use of WFQ, wherever the weights must be changed dynamically. Also, in order to improve transient performance, feedback driven adaptive algorithms must be analyzed and tested for future development.

3.6.4 Example of a WFQ System

A classical example of the WFQ system is the self scaling web server in which many clients request different services. Requests for incoming connections are classified by a SYN classifier, which used for the classification of incoming TCP connections. Based on certain rules, the classifier uses the client's IP address and port number to sort the incoming connections requests into different service classes.

After the required actions are performed, the connection is shiftes from the SYN queue to the accept queue. The system maintains a separate queue for each class, rather than to share same FIFO queue for all classes. Threads perform eviction of the requests from the

queue and in turn threads get queued for the CPU. If the TCP does not accept the queues of the requests, the CPU will become the bottleneck. The WFQ element is introduced into a system in order to perform control of the rate of accepting requests of each class. The operating system's kernel could be modified in such a way that each class is assigned a weight, which determines the rate of accepting requests of that class. Core of the problem is to assign weights efficiently, using the feedback control scheme with a goal to achieve system stability, needed for per class delay in the conditions of the varying input load.

For the weight assignment functioning, the kernel maintains a request frequency, a queueing delay and a request service frequency for each category. The share of resources given to each class is determined by its weight, so for every arrival rate the class queuing delay decreases when they have been allocated a bigger part of the resources, thus making the effect of the resource sharing on the queuing delay becoming less predominant as a class receives more of the shared resources. With each changing of the request frequency, the ratio between the delay and the resource allocation share also changes. This relationship is easily predicted by a queuing theory and it is nonlinear.

The prediction is simple, because of the weight adaptation algorithm, which keeps track of operating points on the functions curve. An operating point of the queue for a class depends on the arrival rate and on the share of resource for the class in question. An algorithm approximates the small segment of the curve around the operating point with a line and this represents the gain of the process. It determines the class delay (change in output) as a function of the change in input (class weight) and the weight of each class is then recalculated for each adjustment. This calculation is based on the approximated linear relationship in such a way that under new weight the delay is set exactly equal as the set point. This algorithm is invoked in every single adaptation interval, which is a fixed already determined quantity.

3.6.5 Transient Behavior of the Feedback Control Loop

While the control loop should converge to the right resource allocation and achieve the desired delay decrease, tests indicate the undesirable interaction between feedback controller and the WFQ scheduler. The feedback control of the WFQ scheduler has two key parameters, which have the most effects on it and need to be kept in mind (the α parameter of the Exponential Wighted Moving Average filter, described in 3.6.6 and the adaptation interval, described in 3.6.7). There is a difference between the theoretical expectations and practical performance. This deviation is explained by an interaction between the scheduler and the controller.

3.6.6 Frequency Noise Filtering

To be able to mitigate the overreaction on the noise in the frequency calculation, the Exponential Weighted Moving Average (EWMA) filter is applied. The EWMA filter is

equivalent to a 1st order low pass filter of an auto regression process. The mathematical function of EWMA can be denoted as follows:

$$\mathrm{F}_r(t) = \alpha \cdot F_{r-1}(t) + (1-\alpha) \cdot \frac{1}{t_r - t_{r-1}}$$

Feedback control theory tells us that EWMA has the tendency to produce lags what slows down the system response. The system slow down depends on α value. Regardless of α being a small value, it allows enough noise to enter the system, so the right value of α should be chosen based on the noise level not on the principle the smaller the better. This decision is connected to the variability in the system load, which in majority of the Internet servers is highly variable, making α value fairly large. A large α, results in the slower control system response. In order to overcome this problem, the separate filter for feedback and the parameter estimation are used, represented in two values of α - the high and low. For the feedback path filter α, a low value is used (α = 0.3). To improve the responsiveness and to improve the parameter estimation and noise reduction the higher α value is set to α = 0.5.

3.6.7 Adaptation Interval and the Impact of the α Parameter in EWMA

The second key parameter of the feedback control loop is the adaptation interval, which is often referred as the sampling period. The adaptation time depends on the EWMA parameter α, which determines the number of the sampling periods that the system needs to settle. If the settling time is perceived in the absolute time units, it is determined by the number of *S* periods, multiplied with the length of the period. In accordance with the valid theory, the fixed α should lead to a smaller sampling period and a faster absolute settling time.

3.6.8 Random Early Detection

The Random Early Detection (RED) algorithm is a commonly used algorithm for congestion control in the TCP/IP protocol. The current TCP transport protocol, which is a most frequently used communication protocol in the Internet, starts with network packet congestion control only after a network packet has been dropped. We can notice, that an initial feedback is required to trigger the RED controlling process. This is the main reason why other more proactive solutions are being pursued. RED function uses two thresholds: (Min/Max redirection thresholds).

- p(F) $\rightarrow$ rejection probability
- f(F) $\rightarrow$ access frequency

Mathematical definition of RED function is defined as following:

Figure 3.1: Graphical Representation of the Random Early Detection Algorithm

$$p(F) = \begin{cases} p(F) = 0 & if F(t) < min.thr \\ p(F) = 1 & if F(t) > max.thr \\ p(F) = \left(\frac{(F(t) - min.thr)}{(max.thr - min.thr)}\right) & otherwise \end{cases}$$

3.6.9 Request Forwarding Mechanisms

Request forwarding mechanisms inform clients about the selection of a replica server, which is generated by the request forwarding algorithms. We will describe the HTTP redirection and URL rewriting mechanisms, which are applied in the Gezora application.

3.6.9.1 HTTP redirection

HTTP redirection spreads information on the replica server sets in HTTP headers. The HTTP protocol permits a web server to answer a client request with a special message, which informs the client to re-submit the request to an alternative server. HTTP redirection may be used with both full-site and partial-site content selection and delivery. The client requests must also be modified to implement a replica server selection. The primary advantage with this approach is the flexibility and simplicity. Another key advantage is that the replication can be managed at a fine granularity. However, the most significant disadvantage with HTTP redirection is the distinct lack of transparency. The overhead perceived through this approach is also significant, as it introduces an extra message

round-trip into the request processing, as well as over the HTTP. The HTTP Redirection is accomplished using the HTTP response code: *307 Temporary Redirect*. This mechanism is illustrated in the Listings 3.1 and 3.2.

Listing 3.1: HTTP Request Header

```
POST /webSite4/Default.aspx HTTP/1.1
Action:"http://130.92.65.130:27246/webSite4/Default.asp"
Content-Type: text/xml; charset="UTF-8"
Content-Length: 582
Connection: Keep-Alive
Cache-Control: no-cache
```

Listing 3.2: HTTP Redirect Header

```
HTTP/1.1 307 Temporary Redirect
Location: http://130.92.65.40:27246/webSite4/Default.aspx
Content-Type: text/html
Cache-Control: private
Connection: close
```

3.6.9.2 URL Rewriting

Despite the fact that most CDN systems use a DNS based routing scheme, many systems use URL rewriting. This system is mainly used with partial-site content selection and delivery, where embedded objects are sent as a response to a client's request. With this approach, the origin server redirects the clients to a number of many surrogate servers by the URL rewriting. Therefore, with a web page containing an HTML file along with embedded objects, the web server modifies references to embedded objects so that the client can fetch them from the best surrogate server. Furthermore, URL rewriting can be:

- Pro-active URL rewriting
- Reactive URL rewriting

In the pro-active URL rewriting, the URLs embedded for objects within the main HTML page are formulated before the content is loaded into the origin server, whereas in the reactive approach, rewriting involves the embedded URLs of a HTML page when the request arrives at the origin server. The primary advantage of URL rewriting is that clients are not bound to a single surrogate server, as the rewritten URLs contain DNS names, which point to a group of surrogate servers.

3.7 Methodologies used for the Gezora Evaluation

3.7.1 Client Satisfaction through Request Time

As we could notice from the Figure 3.2, the time which users are willing to wait for the system response is relatively low. Recent research in Marketing is indicating that the range in costs is three to seven times greater if the company wishes to get a new user instead of keeping the existing one. This information indicates that the system and network performance can have a huge impact on the profit of every company, which is based on enterprise web hosting (cloud services), and that the difference (in the age of the computer technology) between losses and gains for any company is measured in seconds. All other conclusions and comments about latency and speed of request processing in the age of the computer technology are unnecessary. The speed of satisfying the client's request, nowadays, simply equals quality. The content consumer habits (access patterns) are the main concern for the content provider. There are several ways for companies to try to keep the user's attention on the content. For example, if the web page is large and it needs a lot of time to download, a company might choose to put some fast downloading content at the beginning. This will keep the user's attention on the page, until the other content is downloaded in the background. In another scenario the user might be willing to wait more for the desired content, if there is a promise that content will be important to the user. This could be accomplished by better textual or graphic fast loading description of the content. Previously described solutions are representing the attempt to mask the latency of a system or a network but the optimal solution for the content provider is to find a way, which will physically and absolutely reduce the system or network latency and not to try to deceive the user.

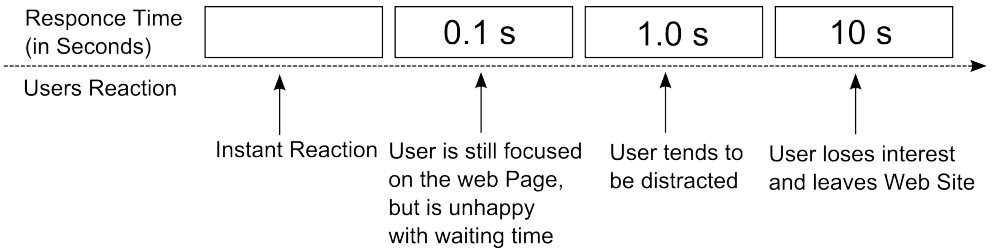

Figure 3.2: How Users Experience the Web Server Response Time

3.7.2 CDNs Performance Measurement Overview

The measurement of CDN performance is undertaken to test its ability for serving of the desired content or service to the end users, at the different request rates. A server has to be able to provide an excellent QoS in every situation, also during the high request rates. It needs to scale very fast and has to be flexible on request rate changes. End users who

utilize the service, need feedback on the performance of the CDN. Performance measurements provide the ability to predict, monitor and ensure the end-to-end performance of a CDN. A measurement is generally achieved by a combination of hardware and software-based probes distributed within the CDN, along with logs from various servers from the CDN. Generally, five principle metrics are used by the content providers to evaluate the performance of a CDN. These are:

1. Cache hit ratio: This can be defined as the ratio of the number of cached documents applied against the total number of documents requested. A high hit rate demonstrates that a CDN is using an effective cache policy to manage its caches.
2. Reserved bandwidth: This is the measurement of the bandwidth used by the origin server. It is measured in bytes and is retrieved from the origin server.
3. Latency: This relates to the end user's perceived response time. A reduction in latency signifies a decrease in the bandwidth reserved by the origin server.
4. Surrogate server utilization: This refers to the fraction of time during which the surrogate servers remain busy. It is a metric used by the administrators to calculate the CPU load, the number of requests served and the storage I/O usage.
5. Reliability: Here, measurements are used to determine the reliability of the CDN. High reliability indicates that the CDN incurs less packet loss and is always available to the clients.

The measurement of the CDN performance can be accomplished based on internal performance measures as well as the customers perspective. This involves the use of internal measurement technologies as well as external services, such as MediaMetrics and Nielsen ratings. However, a CDN provider's own performance testing can often be misleading, as it can perform well for a particular web site and/or content, yet poorly for others. Therefore, to ensure reliable and efficient performance measurement, performance measurement can also be performed by independent third-parties, such as Keynote Systems [53] or Giga Information Group [54].

3.7.2.1 Internal measurement

Internal measurement includes all network monitoring and log analyzing techniques that help the network administrators in finding the potential performance bottlenecks in a network, or to check the node health. The measurement of content delivery throughout the network may be undertaken by collecting and analyzing the logs from the various caches and streaming media servers. All CDN servers can be equipped with the capability to collect statistics to receive end-to-end measurements of its performance. Furthermore, the deployment of probes (either hardware or software) within the network and the correlation of the information collected by probes from the cache and server logs can be used to measure the end-to-end performance of a CDN. The third-party tools can also be used to perform internal measurements and to produce graphical representations of performance data.

3.7.2.2 External measurement

An external performance measurement is an evaluation technique, where the performance measurements and log analysis are accomplished by an independent third-party, usually from within some distance (taking the end user perspective). Usually, the third-party informs the CDN customers of the verified and guaranteed performance. Using this evaluation method, an efficient and independent performance evaluation can be achieved. The computers measure the manner a particular web site performs from the end user's perspective, taking into consideration meaningful metrics on web site application performance as part of the criteria. Giga Information Group [54] is common evaluator, that measures performance by analyzing a number of CDN's parameters during the communication with clients.

3.7.2.3 Network Statistics Acquisition

For internal and external performance measurements, different network statistics acquisition techniques are deployed based on several parameters. Such techniques may involve network probing, traffic monitoring, and feedback from surrogate servers. Typical parameters in the network statistics acquisition process include geographical proximity, network proximity, latency, server load, and server performance as a whole.

3.7.2.4 Network probing

Network probing is a measurement technique where the servers are periodically probed using a ICMP communication protocol. Active probing techniques are sometimes not suitable and limited for some reasons. It introduces additional network latency, which may be significant for small web requests. Moreover, simultaneous server probing can often activate intrusion-detection alerts, resulting in abuse complaints. Probing sometimes may lead to an inaccurate metric as ICMP traffic can be ignored or re prioritized due to concerns of DDoS (Distributed Denial of Service) attacks.

3.7.2.5 Traffic monitoring

Traffic monitoring is a measurement technique where the traffic between the client and the surrogate is monitored to determine the performance metrics. Once the client connects, the actual performance of the transfer is measured. This data is then fed back into the request forwarding system. An example of such traffic monitoring is to watch the packet loss from a client to a surrogate or the user perceived latency by observing the TCP behavior. The response time (latency) is the simplest and commonly used distance metric, which can be estimated by monitoring the number of packets (i.e. traffic) traveled along the route between client and the surrogate. A metric estimation system such as IDMaps [55] measures and disseminates distance information on the global Internet in terms of latency and

bandwidth. This system considers two types of distance information based on timeliness load sensitive and raw (where distance information is obtained considering no load on the network). The estimation of this information is performed through traffic monitoring with an update frequency in the order of days, or if necessary, hours.

3.7.2.6 Feedback from Surrogates

Feedback information can be obtained by periodically probing a surrogate by issuing application specific requests (e.g. HTTP) and taking related measures. Feedback information can also be obtained from agents that are deployed in the surrogates. These agents can communicate a variety of metrics about their nodes. Methods for obtaining feedback information can be static or dynamic. Static methods select a route to minimize the number of hops or to optimize other static parameters. Dynamic probing allows computing round-trip time or other QoS parameters in real time. Network statistics acquisition methods rely on a number of metrics to obtain CDN status information. Geographical proximity is a measure of identifying a user's location within a certain region. It is often used to redirect all users within a certain region to the same Point of Presence (POP). The measurement of such network proximity is typically derived through probing of Border Gateway Protocol (BGP) routing tables. The end user perceived response time (latency) is a useful metric to select the suitable surrogate (i.e. POP) for that user. Packet loss information through the network path is a measurement metric that is used to select the path with lowest error rate. The average bandwidth, startup time and frame rate are the metrics used to select the best path for streaming media delivery. The server load state can be computed based on metrics such as CPU load, network interface load, active connection and storage I/O load. Those metrics are used to select the server with the least aggregated load.

3.7.2.7 Performance Measurement through Simulation

Researchers often use simulation tools to measure a CDNs performance. The CDN simulators implemented in software are valuable tools for researchers to develop, test and diagnose a CDN's performance, since accessing the usage patterns and logs is not easy, due to the proprietary nature of commercial CDNs. Such a simulation process is economical because of no involvement of actual hardware to carry out the experiments. Moreover, it is flexible because it is possible to simulate a link with any bandwidth and propagation delay and a router with any queue size and queue management policy. A simulated network environment is free of any uncontrollable factors such as unwanted external traffic, which the researchers may experience while running experiments in real networks. Hence, simulation results are reproducible and easy to analyze. A wide range of network simulators such as OMNET++/INET [56, 57] are available, which can be used to simulate a CDN to measure its performance. However, the results obtained from a simulation may be misleading if a CDN simulation system does not take into account several critical factors such as the bottlenecks, that are likely to occur in a network, the number of sessions,

that can serve each network element (e.g. router, surrogate server), and the number of traversed nodes considering the TCP/IP network infrastructure.

Chapter 4

Gezora Architecture

4.1 Introduction

Gezora is a server side application which establishes a CDN among a group of interconnected servers. The main role of Gezora is to avoid fast server overloading, caused by sudden traffic spikes. As seen in the introduction, the servers cannot handle the fast growing of requests number due to hardware, software and the communication protocol (TCP/IP) constraints and they need a cooperative environment where they can temporarily outsource one subset of content for a certain time period and redirect the client requests to it. Gezora reacts upon traffic pattern changes before the actual server congestion occurs. This is achieved with the request frequency monitoring Random Early Detection (RED) algorithm, with which Gezora achieves the request-forwarding for a specific content subset, unrelated to the server congestion. The redirection decision is based on two thresholds (min and max thresholds) and a probability function. Using the described mechanisms for request rate monitoring and request routing, Gezora acts as conventional load balancing system, but only at the specific load rates. Generally, the load balancing role is distributed among other servers in the overlay network and it is mixed with the content providing role. The conventional load balancer is a pure centralistic approach, where a fix load balancer (usually one server) is placed between the clients and the server-group and forwards client requests randomly or according some decision rule, whereas Gezora acts on the every server in the CDN and dynamically alternates the role of it, from the content provider to a pure load balancer.

The Gezora Architecture is divided into several subsystems. The main part is the client request frequency monitoring service. The main task for this service is to calculate the request rate frequency and estimate metrics needed for the reactive service. The calculation of the client requests frequency per unit time is accomplished using EWMA, an autoregressive process used in time series analysis. The second part of Gezora is the feedback control and the RED algorithm, a very common algorithm, which is often used for network traffic congestion control, and applied in the TCP/IP protocol. The third part is the request routing (request-forwarding) mechanism, which is the combination of the two

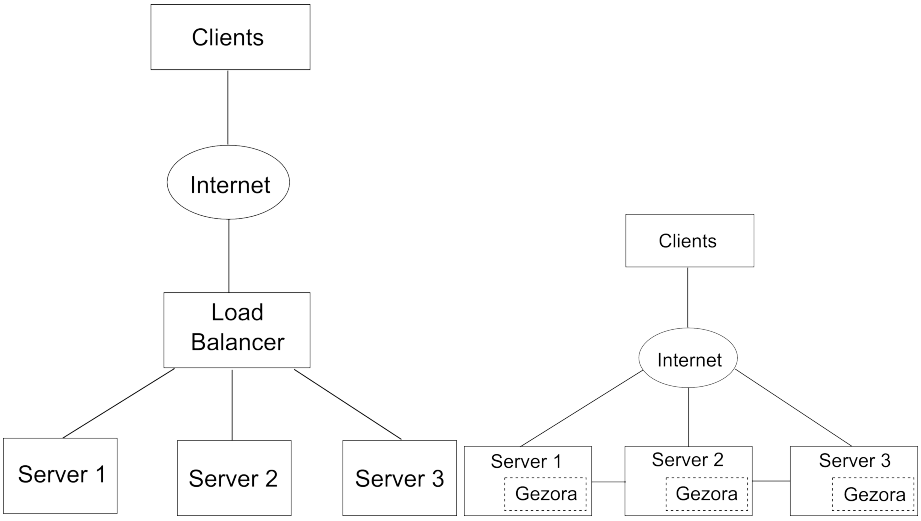

Figure 4.1: Conventional Load Balancer and Gezora Application

request routing mechanisms:

1. HTTP redirection described in 3.6.9.1
2. URL rewriting described in 3.6.9.2

4.2 Deployment of Gezora and its Benefits

The main advantage of Gezora is that it is located already on the servers and that it uses the server's resources. No additional network hardware is necessary, so there is much less work that needs to be done in choosing, buying, setting and connecting equipment with other elements on the network. This clearly is speaking in favor of using Gezora because of the great cost reduction. In this way Gezora is effectively acting as a cheap but functional shield which servers have against flash crowds and system congestion. The physical architecture of a Gezora deployment can be realized in two different ways. Both ways are effective and are transparent to the end user. In the first case, users can be directly connected to the servers on which Gezora is running and their requests will be served or redirected as if they are communicating with a single server (see the Figure 4.6). The advantage in this case is greater parallel computing power and greater bandwidth. In the other case, the end user is communicating physically with one server on which Gezora is running. A drawback in this case could be the bottleneck created by communication through the single server, but this can be overcome by extending the server's characteristics. In real life, a number of servers with weaker characteristics (less computing power and less bandwidth) can perform tasks in same way as the server with better characteristics. As mentioned before, the most important requirement for the end users is that their

requests are served as quickly as possible. It makes no difference if the requests are served by a single server, or a number of servers. The design of Gezora and the use of the overlay network server cloud is providing the end user's computers with much more options and points of access. In this way, the latency is greatly reduced and the client machines are better connected because of the sheer number of connections between replica servers inside the network cloud. If a client's machine is connected to the single replica server in the cloud, it has the ability to access any content on any replica server by the same conditions as if it would be connecting to the cloud via multiple access points.

4.3 Gezora Architecture

The main task of this master thesis was to design and develop a new type of CDN that would minimize the overall latency and response time of existing central web servers and relieve them during the flash crowds. This is achieved using volunteer servers as contributors for the content distribution and load balancing. Volunteer servers are used for temporary storage (caching) of the most frequent requested content, although the server administrators can choose which content subset they want to contribute in the overlay network. The request rates of the web content category will be monitored and the requests will be then forwarded to the surrogates before the actual server overloading occurs. This keeps the whole Gezora CDN in balance and avoids congestion. The Gezora CDN is able to allow volunteer-servers with high bandwidth rate (high upload rate) to support the content distribution even if they can't cache all data. It would allow the request forwarding for specific content category on-demand, which means, a generic proxy caching occurs within smaller space. This implies a higher hit-rate per content category. The content-categorization and replication of specific categories should minimize the overall response time and latency of existing mirror servers. A larger number of servers with small working sets increases the scalability during the large number of requests, because clients will have more optional servers to pick for the same content subset.

A few features make this different from existing proxy servers and CDNs:

- Specialized on mirroring only certain data (category), no generic proxy caching for all data, but generic proxy caching within one category
- On-demand request forwarding
- Clients are not involved in content distribution (no end user implementation required).
- The whole system is transparent to clients (appears as one origin server)
- P2P concepts implemented only on server-side
- The content is not splitted into chunks as in the P2P networks
- URL-based content categorization: splitting a big content working set into content categories. every content category (resource) is denoted with an appropriate URL-address.

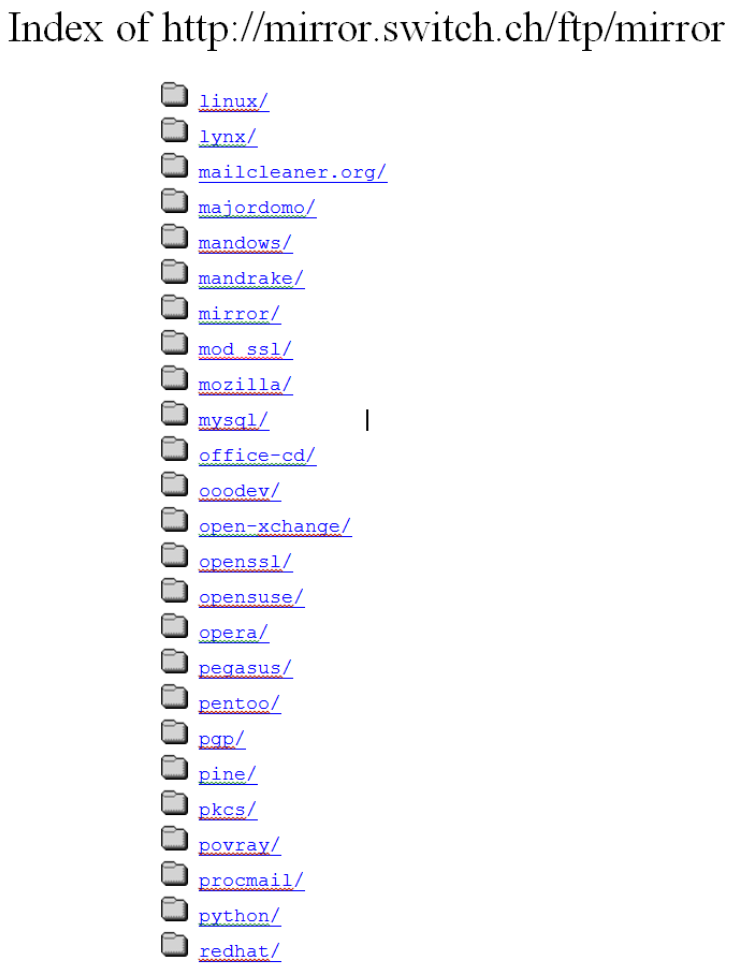

Figure 4.2: Full Directory Structure on the Central Server

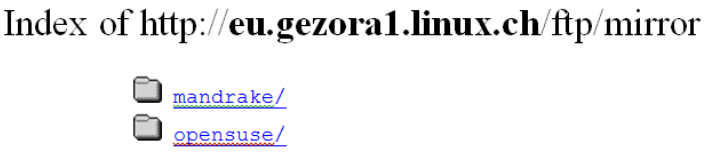

Figure 4.3: Directory Structure for a Specific Content Category on the First Surrogate Server

Figure 4.4: Directory Structure for a Specific Content Category on the Second Surrogate Server

- Creating per category overlay networks with exchanging of messages, which contain a server name with a corresponding resource list and their availability (in terms of load frequency)
- Every unique data source can use a category name as URL address. Only accepted categories will be distributed. Smart per-category limits to allow the administrators to limit resource usage.

The Figures 4.2, 4.3 and 4.4 illustrate the caching of the specific content categories in the URL. Here, we distinguish the host-name of the origin server, denoted in the URL address and the surrogate host names, using the same resource names in the URL address. This method allows Gezora to use fast on the fly URL rewriting mechanism, and redirect clients to other servers within a few milliseconds and before the real request congestion occurs. The information, which is exchanged between servers in Gezora CDN are the host names and the resource path names. With the given resource pathnames, the origin server is able to generate a new URL using the new host name from the surrogate server and the given path name of the resource. For the caching of the replicated content, Gezora reserves some amount of the disk space on the participating servers. The directory structure with the chosen web content for supporting the content distribution will be copied to the reserved disk space. The volunteer servers are now ready to accept any content from the central server, classified for replication with help of the real time monitoring system. This makes the whole content distribution more dynamic and more adaptable to changes. Obviously, the old mirror architecture cannot fulfill this requirement, since it fetches the 1:1 image of the other servers at some fixed time interval. The new system differs from existing P2P networks because it makes a cloud of servers (overlay networks), which is placed between the central mirror server and the clients (See Figure 4.5). To be able to decide, which content category should be replicated, as also when should the chosen category be replicated, Gezora uses request frequency monitoring per content category. If the frequency of a specific content category is above the first threshold, the request forwarding is triggered automatically. With this principle, the decision about content outsourcing is made dynamically, at run time, and Gezora does not require any heuristics (except request frequency per time unit) for the decision about content outsourcing. The decision about content replication and request forwarding is actively controlled by clients, because their request frequency per content category is a crucial parameter for the content outsourcing

and request forwarding.

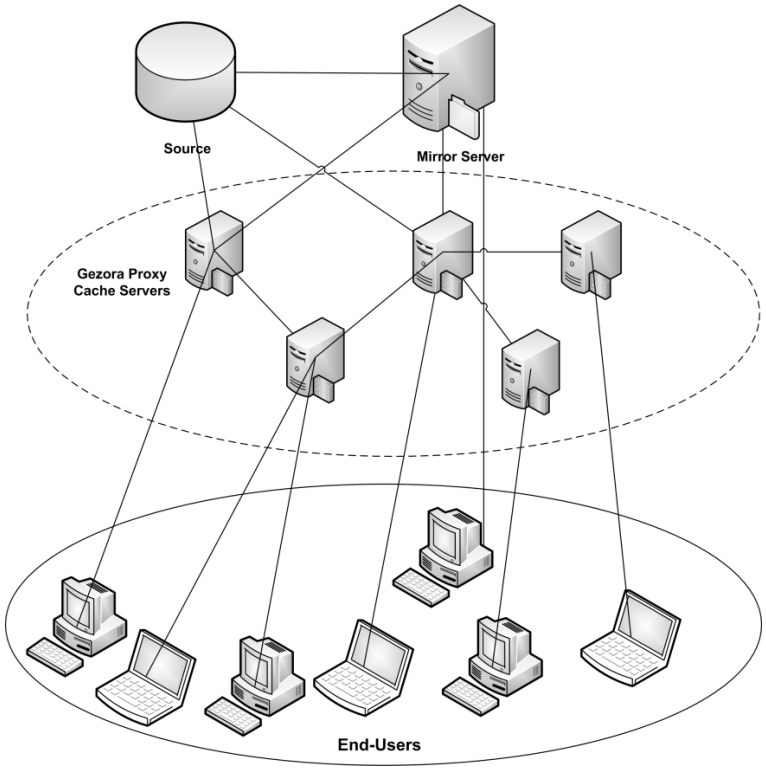

Figure 4.5: Gezora Architecture Overview

The information about the overlay network participants (server host-names), together with the directory structure of the content subset supported on the node is being exchanged between nodes in the overlay network. This information is very important for the efficient request redirection, because the server with a high request rate on one content subset can use any host-name for new URL address generation. As mentioned, the new URL address points to the other source and its replicated content. If a client requests a specific content category, he might be redirected to the surrogate, dependent on the monitored request frequency at that moment. The decision about request forwarding is made using the RED algorithm. Pulling the content from surrogates is fully transparent to end users and is accomplished using URL rewriting. The modified URL is sent back to the clients using the HTTP redirection mechanism, described in 3.6.9. The modified URL address is used for client redirection to the surrogates. This is accomplished within a few milliseconds. Using the described communication process, Gezora provides a loosely coupled network of proxy servers, that allows distributing large amounts of data with little investment for the data provider and every proxy maintainer. At the same time the concept of content categorization in subsets using the URL address hierarchy structure allows to use one surrogate for multiple data sets, reducing the amount of servers and server processes needed.

For example, one surrogate can contribute to the distribution of several resources, which are not located at the same origin server.

4.4 Gezora's Random Early Redirection

We apply the RED algorithm in Gezora to the number of requests per unit time, using an EWMA filter. We must emphasize that the RED algorithm is the core of Gezora's proactivity. EWMA and RED are described in Sections 3.6.6 and 3.6.8. In order to achieve QoS and mitigate overloading, Gezora is redirecting the requests (instead of dropping) to the surrogate (replica) servers using a simple URL rewriting mechanism in combination with the HTTP redirection. For this reason, our algorithm in Gezora is called Random Early Redirection (RER). The principle of RER combined with the EWMA filter, used for frequency calculation, is very efficient and tries to establish a balance in the whole CDN. If the request frequency reaches the first threshold, it will start to redirect the requests to the second server, which will also start to monitor request frequency. If the request frequency at the second server is above the threshold, the second server will also start with request redirection to the third server. This chain reaction tries to use all system resources at once without a big effort for all servers in the group. If the request frequency at one of the servers reaches the second threshold, this server will take the role of pure load balancer and will spread 100% of request to other server, which will also do the same process for his neighbors. Figure 4.6 illustrates load balancing of a Gezora CDN. Here we can see the the client redirection to the two different surrogate servers, dependent on the min/max RER's thresholds.

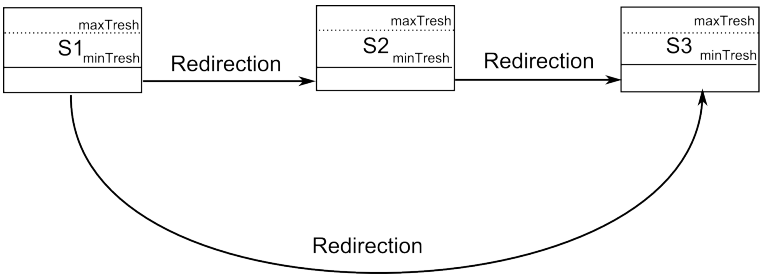

Figure 4.6: Load Balancing on Gezora CDN

On the graph in Figure 3.1 we can notice the mentioned minimum and the maximum thresholds (moments). The minimum moment represents the situation where the access frequency of requests is low as well as the redirection probability. This is natural, because if the need for system's resources is low, the probability that the request will be redirected is also low, and the incoming requests are most probably not going to be redirected. In this case the redirection probability equals zero despite the request access frequency having some, perhaps even more significant value ($F(t) > 0; p(F) = 0$). Unlike the request redirection probability or some other kind of probability, the cache hit ratio can be computed

exactly and it does not contain any degree of probability. Easily mistaken for request redirection probability, it actually represents the system's ability to retrieve the requested content and not the probability that the content will be retrieved. The next scenario is the situation where the access frequency of the requests has increased somewhat and that the redirection probability has also relatively risen. In this scenario, the access frequency and the redirection probability are in direct correlation, since the redirection probability grows proportional to the growth of the request frequency. This is understandable and this is the moment when the proactivity of the system is starting to take effect. At this moment, as the system is getting loaded, some of the requests are timely being redirected in order to prevent the congestion and save the system resources ($F(t) > 0; p(F) > 0$). If we consider yet another jump the in request access frequency, we can notice that the RER's second threshold (max threshold) is reached, and the redirection probability equals 1. In this moment the congestion of the origin server or redirect server is absolute and all requests are being redirected. This effectively means that every single request is being redirected, that all system resources are already employed in some process and that it is 100% sure that all new incoming requests are going to be redirected ($F(t) = +\infty; p(F) = 1$). Since the server cannot send the requested content to the clients and only achieves the requests redirection, the server's role is changed at this point, from the content provider to the pure load balancer. The following Figure 4.7 is displaying the situations and correlations described above.

Bearing this in mind, we can conclude that the redirection probability is directly dependent and a less important factor than the access frequency. This portability of Gezora is greatly increasing Gezora's interoperability, which means that all hardware and software components can cooperate at all levels. In a CDN environment, the interoperability is especially important because of the sheer size of the number of different components: servers, clients and other type of equipment on the network. The interoperability is making sure that clients have the freedom to choose their own type, version and manufacturer of the components and still interact with others on the Internet. The early redirection is significantly reducing the amount of requests, which could overload already well loaded servers. The process of choosing the appropriate surrogate server for clients requests, which are sufficient at some time period, is accomplished using a simple message exchange service between servers. The messages are containing the URL addresses of all content categories, that are supported by a specific surrogate server. The main idea of the random early redirection has two aspects:

- The first part of the idea is to try to redirect requests from flash crowds to the less loaded servers, thus, keeping the origin servers uncongested. This greatly reduces the network latency and enhances the client's experience, perception and satisfaction of using the system.
- The second aspect of the method is randomness. The idea here is to enable the redirection server to use as little resources as possible. This aspect is aimed mostly at increasing the productivity of the server by saving its memory and processor resources. If a redirection server is randomly redirecting the flash crowd's requests,

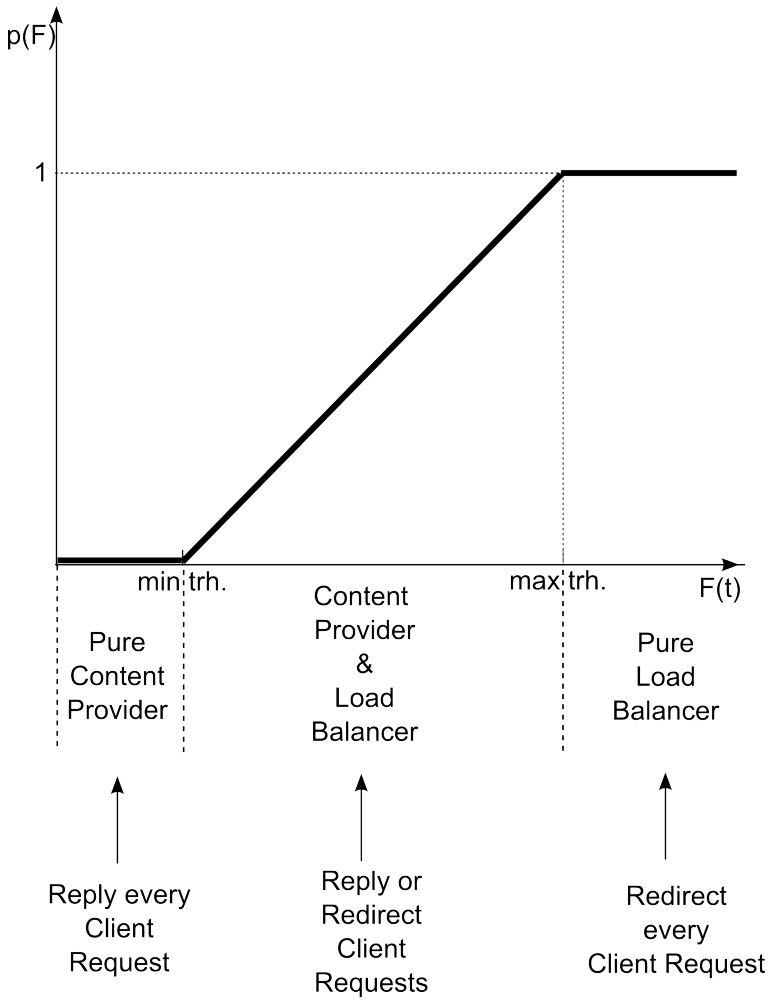

Figure 4.7: Changing Server Roles with RER

it does not have to use the additional memory and processor resources to determine the best or nearest replica server, but it simply passes the request on.

Naturally, the nearest server or the fastest (less loaded) will receive or manage to process the received requests, thus enabling the redirect server to utilize the available resources for more important tasks, or serving the already received requests. This greatly reduces the load on the redirect or origin servers and simplifies the tasks presented to them by flash crowds and sudden surge of requests. In this way, with these two aspects working together, the random early redirection makes the system appear much more powerful and pleasant for clients to use, resolving some of the issues which occur at other CDNs. The random early redirection method is one of the most proactive tools for reducing the network latency.

4.5 Request Forwarding in Gezora

Request forwarding in Gezora is realized using both HTTP Redirection and URL Rewriting mechanisms as described in the Section 3.6.9. Using URL rewriting, Gezora is able to change the server's host name to the host name of the surrogate and send the redirect request to client over HTTP. Basically, the EWMA Filter in the Feedback Control Loop (FCL) is giving the core information: a client request frequency per time unit. This information is used by the RER algorithm. The RER algorithm decides if the normal HTTP response should be accomplished or the HTTP redirection has to be triggered. If the RER's probability value is above the minimum threshold, the request routing, based on the HTTP redirection of the URL-rewriting mechanisms is being activated. Hence, the request forwarding is not occurring the whole time, only if the server load exceeds the first threshold. The probability function is compared with a random number and if a certain condition is satisfied, the request is being forwarded. With growing request frequency, the RER's probability value grows also and the request forwarding probability grows accordingly. In the request routing mechanisms, the URL-rewriting exchanges the central server host name with the one of the surrogate server. The modified URL address is used for the HTTP redirection and to inform the end user about the new source for the requested content. This process is transparent to the end user. The functionality of Gezora's request routing mechanism is shown in Figure 4.8. Here we can see the scenario, where a client requests some content from the origin server(1). It uses a given URL address, which contains also the host name of the origin server. The EWMA based request frequency monitoring is recording the time stamps of occurring requests, and calculates the request rate. Based on the request rate, the RER algorithm decides, if the request forwarding should be accomplished or not. If the request forwarding has to be accomplished, the URL address is being modified with inserting the host name of the surrogate server and sent back to the client using a HTTP redirection code and the modified URL address (2). The client's browser can automatically understand the HTTP protocol codes and can redirect the client to the new source (3). The surrogate servers (new source) are retrieving the content to the client (4).

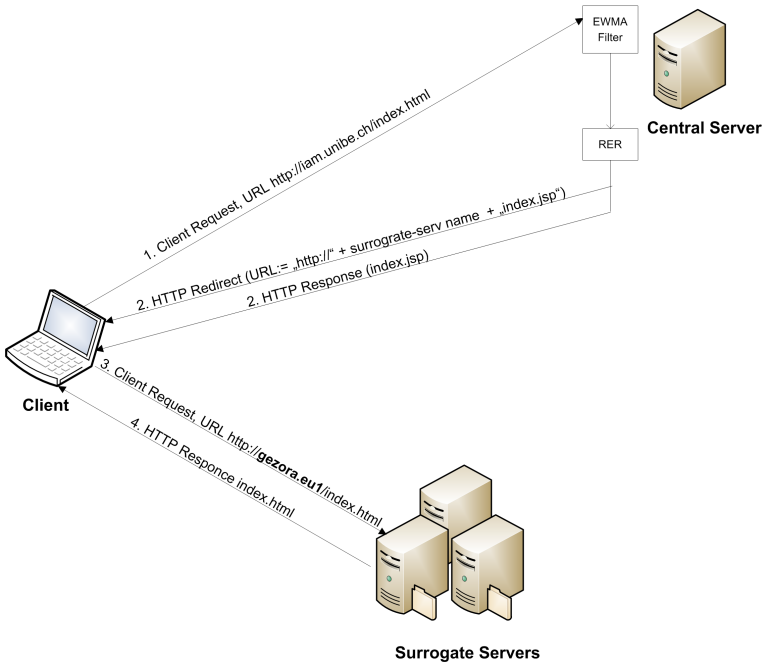

Figure 4.8: Gezora's Request Forwarding Mechanism

Chapter 5

Gezora Prototype Implementation

5.1 Introduction

Over the last several years, Java has become the predominant programming language for server-side programming. The main reason for this trend is that the Java Servlet Application Programming Interface (API) provides a standardized method for enabling web servers to provide support for dynamic content generation. Nowadays, servlets are a natural choice as a development platform for web programming. In this environment, a user is making the request to a web server, which invokes a servlet designed to handle the particular request. The servlet fulfills the request and returns a response to the client. The fact that the servlet is dealing with a single request at a time ensures the exclusivity of system resources for that request. This is a guarantee that the client's request will get the best possible treatment and access to the required resources. While the mere concept of servlets may seem similar to other technologies for dynamic content, they demonstrate several important advantages. Servlets are persistent in memory between invocations, which improves the performances significantly. Furthermore, servlets are portable across operating systems and servers and they have the access to all APIs of the Java platform, which makes it easy to create a servlet, that interacts with a database. The portability of the servlets has greatly increased its interoperability, which allows all hardware and software components to cooperate at various levels. This interoperability of servlets is especially important in an environment with a large number of different components, such as a CDN environment. The interoperability is making sure that clients have the freedom to choose the type, version and manufacturer of the components and still interact with others on Internet. All these characteristics have contributed to a very fast spread of the servlet technology on the World Wide Web. For this reason, we chose to implement the first Gezora prototype using J2EE and Java Servlets. In addition to the first Gezora prototype, we implemented also a second Gezora prototype, which has a different architecture. It consists of the same algorithms used in the prototype 1 (FCL and RER), but instead of applying the algorithms on the HTTP protocol layer, we applied them on the Ethernet packet layer (Layer 2). The main idea was to intercept the raw Ethernet packets and to search for the HTTP requests of the specific content category. This would trigger the HTTP redirection on the network

layer 2. Due to using the C++ programming language for the implementation, this whole concept offered several performance benefits, most importantly for the traffic monitoring and reactive algorithms. Another important aspect was the fact that this approach did not interfere with the big Garbage Collector overhead of Java. Nevertheless, the implementation of this prototype imposed a few challenges regarding the complexity of the TCP/IP protocol and further development.

5.2 Design Overview: First Gezora Prototype

As mentioned in Section 4.3, the main application of Gezora is the content categorization in the overlay network to minimize the overhead for the content replication. Namely, when a client's request is received, the time stamp of the reception is being calculated and used as a parameter for the frequency calculation with EWMA, described in Section 3.6.6. For the request frequency calculation with EWMA filter, two requests with different time stamps are required. The request frequency calculation is accomplished using a mathematical expression defined in Section 4.3. The request monitoring system is tracking the request frequency and when the conditions in the RER algorithm are met, the request is being forwarded to the surrogate using a request routing mechanism (HTTP redirection and URL rewriting). If the request frequency values are indicating that there should be no congestion in the near future, the client's request is being served and the desired web content is returned to the client. If the request frequency of the one web content category is growing very fast, it is indicating a partial congestion within one web content category (request frequency monitoring and RER are active for every content category, which is supported by surrogates). This should not mean that the whole server system is congested. The server allows low request frequencies at the other web content categories. At the content category where the potential congestion is being indicated, the request is redirected out of the system to a surrogate server. Because the content could be dispersed on many replica servers in the overlay network (server cloud), all required content could be retrieved from one or a number of replica servers in one or several iterations. End users are able to perceive all these processes only in the form of a greater or a lesser latency as the difference in the content retrieval from one and several replica servers remains undetectable to them. The figure 5.1 illustrates different processes within Gezora CDN, which are set in motion upon a client's request.

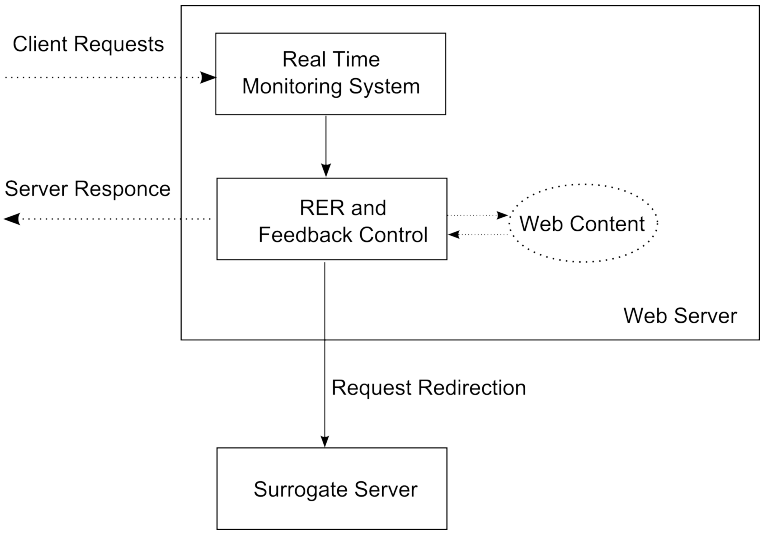

Figure 5.1: Content Categorization Overview

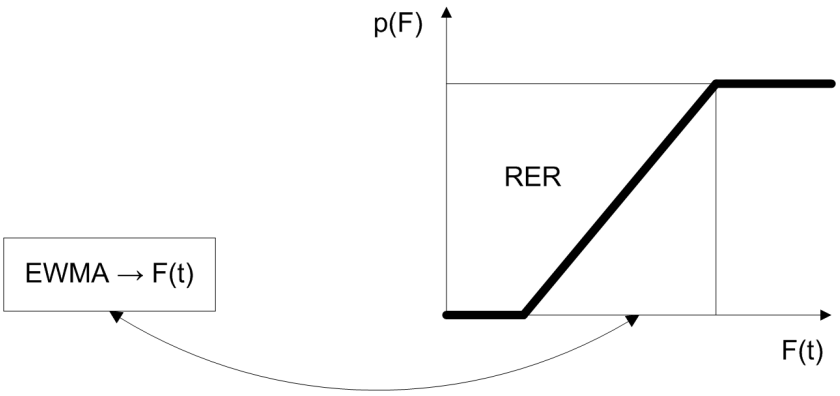

Figure 5.2: EWMA Filter's Frequency Applied to RER

5.3 First Gezora Prototype Implementation

The first Gezora prototype is implemented using Java servlets technology, described in 3.5. It typically runs on the server side as a background process and tracks the client requests time using Java servlets. The core of the Gezora prototype is a Java based server implementation, better known as Apache Tomcat Server [47]. Since the Apache Tomcat Server is positioned in the application layer (network layer 7), the Gezora prototype also acts in the same network layer. Figure 5.3 illustrates the position of the Gezora prototype in the ISO/OSI network layer architecture.

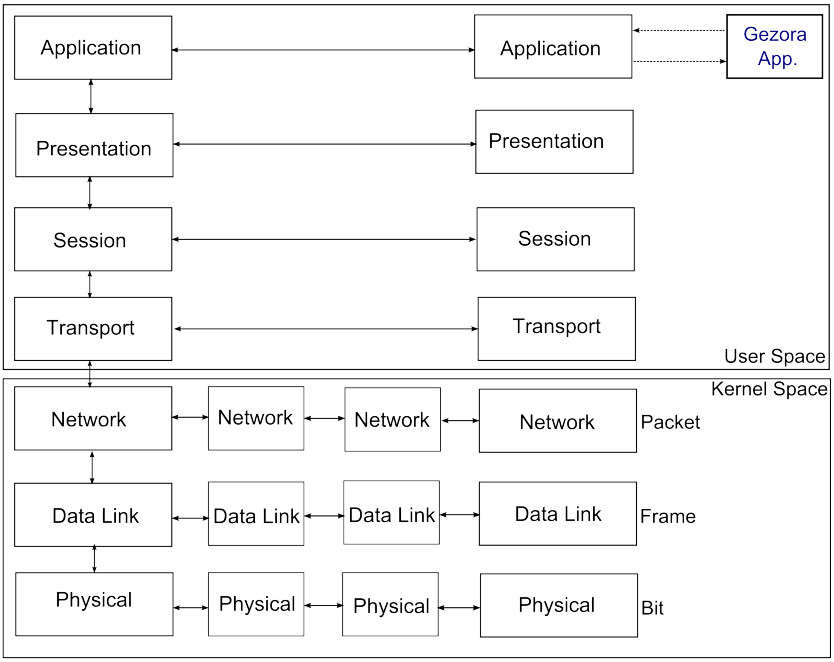

Figure 5.3: First Gezora prototype in the ISO/OSI Network Diagram

Other Gezora components, like EWMA filter, RER and request routing mechanism are implemented in Java and bound to the Java servlets. Java servlets are very important for the request forwarding implementation, since they have already implemented methods for HTTP redirection and URL retrieval. Figure 5.4 shows the structure of the Gezora implementation in the Unified Modeling Language (UML), which consists of the following implementation elements:

1. Servlet: The core of the Gezora implementation with Java methods for HTTP request/response handling
2. Random Early Redirection: Request Redirection Logic
3. Access Meter: Request Frequency Monitoring System

4. Util: Additional Methods

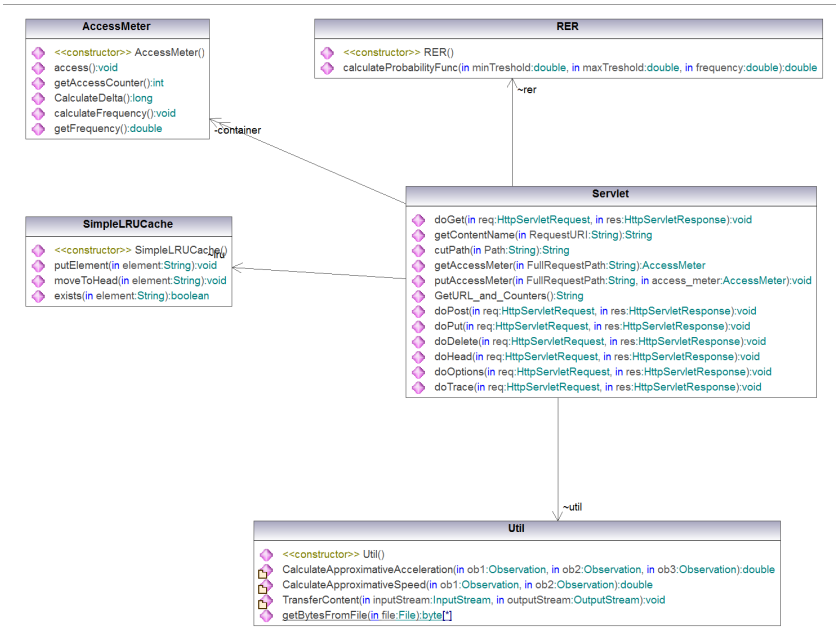

Figure 5.4: The Structure of the Gezora Prototype Implementation as UML Diagram

5.3.1 Servlet

A servlet class is the core of each Gezora prototype implementation. It contains all methods necessary for the client's HTTP requests and server response handling. Thus, servlets allow us to use the *HttpServletRequest* and *HttpServletResponse* classes as arguments in *doGet* Java method and apply their methods for URL retrieval of the requested content, HTTP redirection, content transfer, etc. For the description of the HTTP protocol, please refer to the RFC on ietf.org.

The first step in the communication process on the server side is to get an URL address of the requested content as a string. In response to the client's request, the full URL path is being constructed with the concatenation of two strings: RequestURL and RequestURI. In the next step, the *AccessCounter* class is generated for the request frequency calculation. A new instance of the *AccessMeter* class is generated for every client request. This class is then added in one container, using the full URL path as a key. In the next step, a random number is generated and compared with the value calculated with the RER algorithm. If the generated random number is smaller than the RER value, the so called redirection flag is set, which will be further used for indicating the client's redirection. On the other hand, if the generated random number is higher than the calculated value from the RER algorithm, the request frequency is calculated using the *AccessMeter* class.

This mechanism allows us to avoid the unnecessary frequency calculation, even in case of redirection. If the redirection flag is set, the host name of the origin server is exchanged for the host name of the surrogate server in the full URL path (URL rewriting), and this modified URL is used as a parameter for the HTTP redirection. The HTTP redirection is implemented in Java servlets. The method *sendRedirect* initiates the HTTP redirection, using the new URL address as argument. In case of the lack of redirection, the content is transferred to the client, using the data stream as byte array. This can be done using the *write* method in the *HttpServletResponse* class. The listing 5.1 shows the described Java methods and algorithms, applied in the communication process, during one client request.

Listing 5.1: Java Servlet Get Method in the Gezora Prototype Implementation

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{

 FullUrlPath = req.getRequestURL() + req.getRequestURI();


 if(req.getParameter("reqType") != null)
 {
        if(req.getParameter("reqType").equals("client"))
        {


     System.out.println("Client");

        synchronized(this)
        {
          if(this.getAccessMeter(FullUrlPath) == null)
    {
                    this.putAccessMeter(FullUrlPath, new AccessMeter());
    }

          probability = rer.calculateProbabilityFunc(minTreshold,
                maxTreshold,
                this.getAccessMeter(this.FullUrlPath).getFrequency());

          if(random.nextDouble() < probability)
          {
                  redirect = true;
                }else
          {
            this.getAccessMeter(FullUrlPath).access();
          }
 }
        if(this.redirect)
  {
          clientAddress = req.getRemoteAddr();
          clientPort = req.getRemotePort();

          String localAddress = req.getLocalAddr();
          int localPort = req.getLocalPort();
```

```
          res.sendRedirect(res.encodeRedirectURL
                                ("http://" + surrogate.getInetAddress
                                (closestNodeAddress).getHostName()) +
                                ":" + req.getLocalPort() + req.getRequestURI()
                                + "?reqType=client");

          this.redirect = false;
       }else
       {

             tempReqPathInfo = req.getPathInfo();
inputStream =
this.getServletContext().getResourceAsStream(req.getPathInfo());

  if (inputStream != null)
  {
     System.out.println("Transfering Content to Client");
     util.TransferContent(inputStream, res.getOutputStream());
     if(lru.elements.contains(this.getContentName(req.getRequestURI())))
     {
       lru.moveToHead(this.getContentName(req.getRequestURI()));
     }
                else
   {
       lru.putElement(this.getContentName(req.getRequestURI()));
   }

 }else
  {
    System.out.println(req.getLocalName().toString());
    if(req.getLocalName() !=
                this.getServletContext().
                getInitParameter("masterServer"))
   {
     File file = new File("");
     System.out.println(this.cutPath(file.getCanonicalPath()));
     FileOutputStream out =
     new FileOutputStream(new File(this.cutPath(file.getCanonicalPath()) +
     "/apache-tomcat-6.0.18/webapps/SimpleServlet/" +
     this.getContentName(req.getRequestURI())));

      int readBytes;
      byte[] byteArray = new byte[2048];

      System.out.println("connecting to MasterServer");
      url = new URL("http://" +
                this.getServletContext().getInitParameter("masterServer") + ":"
                + req.getLocalPort() + req.getRequestURI()+ "?reqType=server");
      urlConnection = (HttpURLConnection)url.openConnection();
      InputStream iStream = urlConnection.getInputStream();

                while((readBytes = iStream.read(byteArray)) > 0)
                {
                        out.write(byteArray, 0, readBytes);
                      }

                lru.putElement(this.getContentName(req.getRequestURI()));
                File fl = new File(this.cutPath(file.getCanonicalPath())
                      + "/apache-tomcat-6.0.18/webapps/SimpleServlet/"
                      + this.getContentName(req.getRequestURI()));
                 res.getOutputStream().write(util.getBytesFromFile(fl));
```

```
                    iStream.close();
        out.close();

            }
          }
        }

          }else if(req.getParameter("reqType").equals("server"))
          {
            System.out.println("Server");


            inputStream = this.getServletContext().
                 getResourceAsStream(req.getPathInfo());

        if (inputStream != null)
        {
          System.out.println("Transfering Content to Server");
          util.TransferContent(inputStream, res.getOutputStream());
        }else
        {


        if(!req.getLocalName().equals(
                        this.getServletContext().getInitParameter("masterServer")))
       {

          File file = new File("");
          System.out.println("Canonical Path: " + file.getCanonicalPath());
          System.out.println("Absolute Path: " + file.getAbsolutePath());

          FileOutputStream out =
          new FileOutputStream(new File(this.cutPath(file.getCanonicalPath()) +
          "/apache-tomcat-6.0.18/webapps/SimpleServlet/"
          + this.getContentName(req.getRequestURI())));
            int readBytes;
            byte[] byteArray = new byte[2048];

          url = new URL("http://" +
          this.getServletContext().getInitParameter("masterServer")
+ ":" + req.getLocalPort() + req.getRequestURI() + "?reqType=server");
          urlConnection = (HttpURLConnection)url.openConnection();

                        InputStream iStream = urlConnection.getInputStream();

                  while((readBytes = iStream.read(byteArray)) > 0)
                  {

                        out.write(byteArray, 0, readBytes);
                        res.getOutputStream().write(byteArray, 0, readBytes);
                  }

                  inputStream.close();
                  iStream.close();
                  out.close();
      }
}
  }
 }
}
```

5.3.2 Access-Meter

The Access-Meter is a Java implementation of the request frequency calculation with the EWMA filter (See Figure 5.6). For this purpose, a time-stamp of the incoming client request is used for the request frequency calculation. In response to a client request, a time-stamp is recorded and used as a first parameter for EWMA calculation. Since the frequency cannot be calculated using only one parameter, the EWMA calculation requires another client request for request frequency calculation using an EWMA filter and one frequency parameter from the previous time period. With the next incoming request, the new time-stamp is recorded and used as the second time-parameter for the EWMA calculation. Since the EWMA filter needs one old frequency sample, the time-stamps from the first two client requests need to be recorded in order to be used for the frequency calculation, which will in turn be the base for the next frequency calculation. This implementation is based on the principles of mathematical induction. For the simplicity reasons, the initial frequency equals 0. This frequency is applied for the time period t[r-1]. Started with the value 0, the first two client requests are captured by the Java servlets and their time-stamps are recorded. Subtracting the two time-stamps allows the tie gap between those two client requests. The reciprocal value of the calculated time gap is then multiplied with the constant α and added to the old time period frequency value, which is subsequently multiplied with the second constant (1-α). The Figure 5.5 illustrates the request frequency calculation with EWMA using Java servlets and recorded time-stamps as time parameters. Code listing 5.2 denotes the Java implementation for the EWMA-based request frequency calculation.

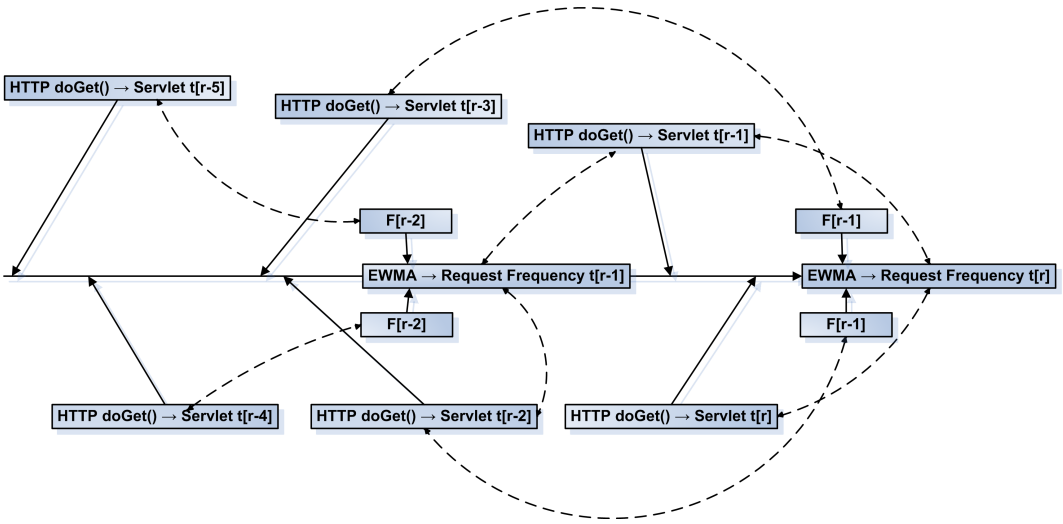

Figure 5.5: EWMA Access Frequency Calculated using Java Servlets

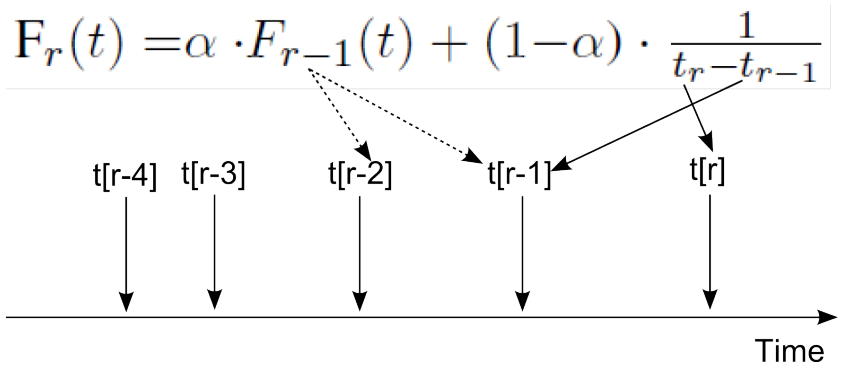

Figure 5.6: EWMA Filter

Listing 5.2: Access Meter

```
public class AccessMeter
{
  int accessCounter;
  long time;
  double frequency;
  long last_time;
  double last_frequency;


  public AccessMeter()
  {
        this.accessCounter = 1;
        this.frequency = 0;
        this.last_time=0;
  }



  public void access()
  {
    accessCounter++;
    this.calculateFrequency();
    this.last_time = this.time;
    this.last_frequency = this.frequency;
  }

  public int getAccessCounter()
  {
    return accessCounter;
  }

  public long CalculateDelta()
  {
          return (this.time = System.nanoTime()) - this.last_time;
  }

  public void calculateFrequency()
  {
```

```
          double newFreq = getFrequency();
          this.frequency = newFreq;
  }

  public double getFrequency()
  {
long delta = this.CalculateDelta();
double newFreq = 0.5 * this.last_frequency +
0.5 * (1000000000.0/delta);
return newFreq;
  }
```

5.3.3 Random Early Redirection

This implementation features the RER algorithm as a function of three parameters. The first parameter is the request frequency, which is used for comparison of the thresholds of the other two parameters. The RER functioning principle is very simple. If the frequency is above the first threshold, a server changes its role from the pure content provider to the semi content provider and semi load balancer. The request frequency value is subtracted by the first threshold and divided by the difference of the first and second thresholds. This value is then compared with the probability number, generated with the Java random number generator. If the probability is smaller, it indicates that the client request has to be redirected, in order to be served by a surrogate. If the request frequency value grows further and is above the second threshold, the server's role is changed from the mixed, to the pure load balancer. The return code 1 indicates the client's redirection. If the frequency starts decreasing, the server's role is changed again to the mixed mode and vice versa. This operating mode assures the flexible changing of the server roles in the overlay network and self-adaptivity, as well as fast reaction to the peak client request rates. Code listing 5.3 denotes the Java implementation of the RER algorithm.

Listing 5.3: Java Code for Random Early Redirection Implementation

```
double calculateProbabilityFunc(double minTreshold, double maxTreshold,
double frequency)
{
  if(frequency < minTreshold)
  {
    return 0;
  }
  else if(frequency > maxTreshold)
  {
     return 1;
  }else
  {
    return (frequency-minTreshold)/(maxTreshold-minTreshold);
  }
}
```

5.3.4 Request Forwarding

The request forwarding implementation is accomplished using a simple URL-rewriting mechanism, combined with the HTTP redirection mechanism, implemented in Java servlets. For the content name retrieval the Java servlet methods: *getURL()* and *getURI()* were implemented in the *HttpServletRequest* class. Namely, the origin server as well as other servers in the overlay network exchange the host names of their neighbors periodically and use them for the request forwarding. If a new volunteer server establishes the connection with a Gezora CDN, the name of this host will be recorded and forwarded to every node in this network. The server administrator has the possibility to choose the size of the hard-disk used for contributing to the content distribution, as well as the supported content. The URL names of the content have to be standardized and the same goes for every volunteer server. As soon as the volunteer server has this content, his host name will be effectively used as a surrogate in case of a sudden peak in the request frequency. The host name is easily replaced in the URL address, and the modified URL address can be sent to the client, using a HTTP redirection.

5.3.5 Content Transfer Implementation

Gezora's Util class comprises a method, which is responsible for the content transfer from a server to the client and also to other servers. This method transfers the content as a data stream using a *byteArray* buffer and *write()* method, implemented in HTTP response class. The implementation and application of the content transfer can be seen in the code listing 5.1.

5.4 Second Gezora Prototype Architecture and Design

The goal of the Second Gezora prototype implementation was to improve Gezora's performance and to avoid certain bottlenecks, which arise using the Java programming language for implementation on the application layer. The main idea behind this prototype was to parse and filter Ethernet for HTTP request containing packets in the payload and to destroy them, in case of redirection. Destroying the sufficient Ethernet packets would initiate the artificial generation of Ethernet packets, which contains the modified HTTP header with the redirection code and modified URL address. Those packets are sent to the client who initiated the HTTP request. The request frequency monitoring and random early redirection are subsequently implemented on the second network layer (See Figure 5.7).

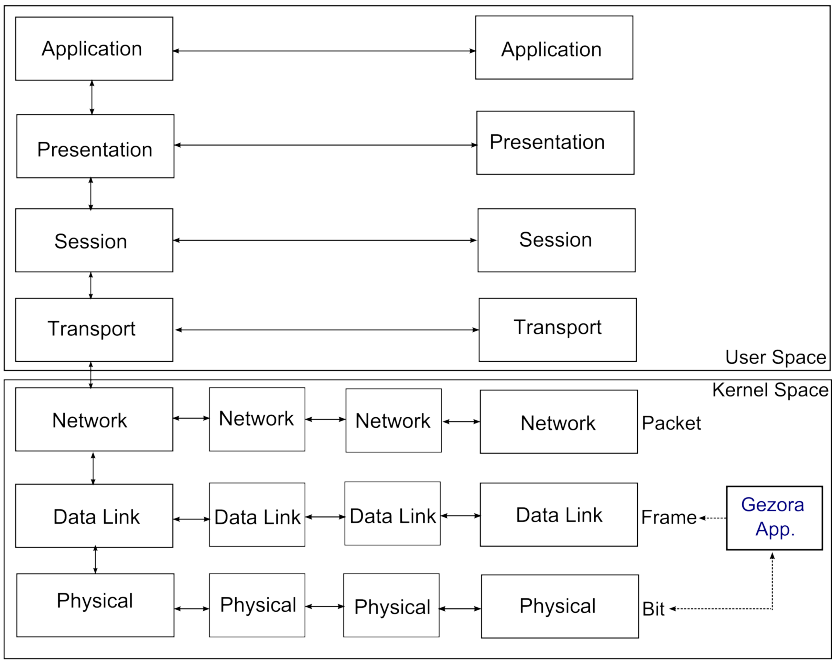

Figure 5.7: Second Gezora Prototype in the ISO/OSI Layer Diagram

Using this approach, the client would be informed about temporary request redirection, as with the HTTP protocol, however the whole process would proceed much faster and would spare server resources, since this system does not allow the Ethernet packets to be transferred to the higher network layers. The figure 5.8 illustrates the flow occurrence during the Ethernet packet monitoring and parsing.

5.4.1 Request Forwarding

The Ethernet packets intended for sending to the upper network layers would typically be destroyed, whereas a new Ethernet packet would be generated with injecting of the artificial generated HTTP header, which contains the HTTP redirection code with the new URL address from the surrogate server. This packet would be sent back to the client, and would initiate a HTTP redirection. This concept would benefit more in performance since the redirection is accomplished earlier (on Network Layer 2). An additional advantage of this prototype would be the lack of the Java garbage collector, which is active in the first prototype using Java servlets.

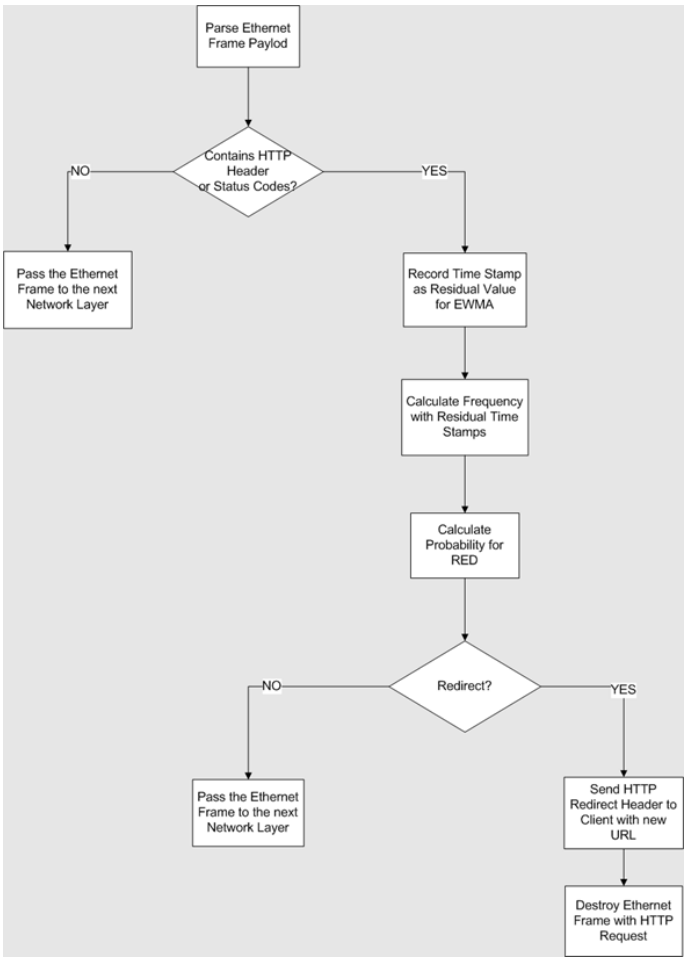

Figure 5.8: Flow Diagram

5.4.2 Disadvantages of the Second Gezora Prototype

Besides numerous advantages, this approach also imposes new challenges, which are exhibited in the lack of packet synchronization (data and ack) in the TCP/IP network protocol. Hence, the server response synchronization was not able to be carried out in the same way, as in normal TCP during the HTTP communication. Basically, during the establishment of TCP connection, the session is being starting with a three way handshake mechanism, and controlled by the following TCP flags:

- SYN = Synchronize
- ACK = Acknowledge
- PSH = Push
- URG = Urgent
- FIN = Final
- RST = Reset

The TCP session establishment can be described as follows:

1. A client sends a TCP Synchronize packet (SYN) to a server
2. A server receives the client's SYN packet
3. A server sends a Synchronize-Acknowledgment (SYN-ACK) to the client
4. A client receives a servers's SYN-ACK and sends ACK back to a server
5. A server receives ACK and the TCP connection is established

The TCP SYN packet is sent to the recipient to allow the TCP session establishment. This consent is indicated by a TCP SYN-ACK packet. The problem arises during the capturing of Ethernet packets and searching for the HTTP request. Assume the HTTP request is found and a request forwarding initiated. TCP session will not be closed properly, and the TCP SYN packets will be continually sent to the client, and SYN-ACK's are received, even if the request forwarding to the surrogate server already occurred. This leads to a serious interference in the working capacity of server (and client also), and can lead to a system crash. For this reason, the second prototype was not considered for further performance analysis of Gezora.

Chapter 6

Gezora Prototype Evaluation

6.1 Introduction

As previously described in chapter 3.7.2.5, the evaluation of the Gezora Prototype was accomplished using the traffic monitoring evaluation, which gives a valuable feedback regarding the Gezora prototype implementation performance. For this purpose, the following evaluation models were applied:

1. Java Client Model (Used for RER Evaluation)
2. Web Stress Tests (Used for Gezora Evaluation)

The Java client model applied for evaluation of the RER algorithm allowed us an in-depth insight into the RER implementation behavior implemented in the Java programming language. Using this approach we were able to characterize the RER's behavior, which influenced the flexibility and on-demand scalability of the systems with the minimal resources. The first evaluation scenario included employing one origin server and several nodes for the client simulation. The client model was implemented in Java programming language, whereas for simulating the client requests, Java servlets were used.

The second form of evaluation was performed using web server stress tests from Paessler [58], which were focused on Gezora CDN. This network monitoring application simulates real-life client requests and applies them directly to the origin server using a given URL address. The client's request frequency can be parametrized and changed during the evaluation time period.

The following sections will give a brief overview of various CDN evaluation methodologies followed by a more detailed description of employed evaluation models. Finally, the Gezora evaluation results will be presented.

6.2 CDNs Evaluation Methodologies Overview

6.2.1 Important Parameters for CDN Evaluation

One of the most important parameters influencing the stability and performance of the CDN system is the cache hit ratio, a metric describing the frequency of a successful response to a client request and timely delivery of the requested content. The cache hit ratio is directly connected with the system design and it indicates the excellence with which the system design is performing the task for which it is designed (content distribution). The effectiveness of the cache hit ratio is directly indicating the design quality and it represents one of the most important system characteristics. Every origin server has a designated bandwidth, determined either physically or logically by content providers and network engineers. For the network bandwidth to be optimally used as a network resource, periodic analysis is needed in order to set the appropriate amount of bandwidth for the needs of users and to retain the lowest possible costs for the content providers. If the bandwidth is greater than the client's needs are, the content providers are loosing money and paying for something that is not fully utilized. If the bandwidth is lower than the client's needs are, the content might not reach all users in need and low bandwidth would significantly decrease the experience of using the system. Having this in mind, the engineers and programmers need to periodically analyze bandwidth utilization, which is tightly connected to the issue of bandwidth management. Different algorithms are entrusted to manage the bandwidth in an automatic manner.

6.2.2 Network Traffic Analyzing and it's Benefits

Network traffic should be analyzed periodically both in the volume and in the structure. If the findings show that the algorithm is handling all requests from flash crowds and other redirection needs in a satisfying manner, it can be concluded that the algorithm is handling tasks satisfactorily and that the additional adjustments are not necessary. If the bandwidth is dedicated logically to a server and if the comprehensive network analysis shows the frequent need for a higher bandwidth, the adjustments in the network software could be made in order to allocate more bandwidth to the server. With time, engineers and content providers could even start acting proactively and allocate more network resources to more popular content and thus prevent congestions in advance. If experience shows that some content is highly popular, more bandwidth could be allocated to these particular servers. On the other hand, an analysis of the traffic structure could prove to be more complex. For example, if such an analysis suggests that more popular content should be packed, made available in some other format, divided into smaller parts or copied to different servers, which are more appropriate for storing particular content types. These forms of analysis would contribute to adjusting the optimal bandwidth consumption for the users and content providers. Good bandwidth management and careful planning of the system design can reduce the client perceived latency to the minimum or at least to an acceptable level. Nevertheless, in certain cases the system or network latency can impose

a major obstacle. From the viewpoint of the average user, it makes no difference whether the latency has occurred on the server or somewhere in the network or a node. Because of this, we will not insist on a distinction between these two sources of latency and will address the latency problem regardless of the source of origin in the following text. In the first case due to latency (regardless of its source) the users may loose patience and give up on attempts to download certain desired content. In the second case, the system could be so congested that the distribution of the content could be impossible even if users have the patience to wait for the download. In the first case, which is easier to resolve, the solution could be to mirror the content on the other server which has less congestion and in the other case, little could be done without physical intervention or natural request dissipation. The utilization of surrogate servers is not something that the average user would be much concerned about directly. End users are only interested in the latency as a whole being as minimal as possible and not how it is achieved, and how the network resources are organized and utilized.

6.2.3 Surrogate Utilization Monitoring

The monitoring of the surrogate server utilization is helping the network administrators to plan more precisely the network design and network resources. Accurate planning ensures distribution of content with the minimal latency and optimal usage, once the system is deployed in a real life situation. Optimal utilization of all resources can be sometimes very important in three aspects: user satisfaction, content provider's cost reduction and importance for interest group in general. A lot has been said about the importance of user satisfaction so it is needless to say that it is the core reason for all the efforts. Any content provider has the interest to enable their content to be available to the end users. The motivation can range from profit making to information sharing in order to achieve social prestige and spreading of ideology. Sometimes, in certain cases it could be important to the local community or an interest group to get much needed information not regarding profit generation. For example a university professor might need to distribute materials for exam preparation, a football club might want to distribute content to the fans, etc. All this is pointing to the importance of easy and fast content distribution, for which all sides have interest in (content providers, network administrators and end users). All measurements (cache hit ratio, request time, reserved bandwidth, latency, surrogate server utilization) are important, but if the reliability or the network or a system is not satisfying the main requirement, all previous factors are worthless. It makes not much difference if the latency and server load are low, if cache hit ratio and bandwidth are high if the system is not on line 24/7 or if the connection is breaking often. The most important requirement is to provide the same performance to the end users, regardless which factor is influencing the system.

6.2.4 User Surveys and a Real-life Evaluation Methods

In order to get the performance information of the system, network administrators need to monitor all server characteristics and set parameters for the optimal server performance. Both, the content providers and end users share the same need for optimum performance from the system and network, regardless how it is achieved. Independent third parties can utilize user panels (predetermined groups of users) for conducting user surveys, which can be selected over and over again, each time the survey is being performed. The importance of each survey lies in the wide number of tests performed with a large number of users. This evaluation method is optimal, since it corresponds to the real-life environment. The main drawback of this evaluation method is a long time period for results collection. Usually, we need to get the evaluation results as fast as possible. To be able to accomplish this requirement in a short time period, we usually use mathematical models to simulate real life situations. Nowadays a very powerful mathematical models could be employed to analyze the system performance and model a possible future behavior based on the gathered data. Environment simulation and computer models although very accurate, still cannot substitute a real life testing. Hardware testing is usually performed by manufacturers whereas the network component testing is done mainly by network administrators. Whereas the software testing can be automated with unit tests, the main difficulty is to define initial requirements before the software engineering and implementation starts. This is a rather challenging task, since a large number of parameters are influencing the implementation. In certain cases it is more efficient to apply real-life tests using beta versions. The probability of one user finding a use case, which is not covered, is higher than when a software engineer tries to locate the same use case. Third party guarantees by its name the results of the testing and provides trustworthiness for both the users and system creators. In this way users have more confidence in the system and its performance and the system creators can achieve a great marketing advantage just by simply employing third party for system testing.

6.2.5 Conclusion and the CDN's Evaluation Methods Comparison

Although network statistics acquisition is complex and important, it is not less complex or important than data analysis. Data analysis, such as access pattern, bandwidth usage and response time examination, usually requires employment of teams of experts, which include information technology experts as well as the experts in other fields such as statistics, mathematics, social science, behavioral experts etc. Due to reasons listed above, network probing must be approached with care and the results are ought to be analyzed with great care. It is up to the network administrators, system administrators and end users to choose which parameter is the most important to them but all values of parameters are indicating the state of the system as a whole.

6.3 Java Client Model and Gezora's RER Evaluation

The Java client model was initially implemented for the RER Evaluation, but it also gave the basic idea about the evaluation concept of the Gezora system. This simple core tests are aimed at testing the basic system functionality and the responsiveness of the RER algorithm. If a basic system functionality is flawed, there is no reason to test the whole system with more complicated tools.

In the evaluation scenario, we are trying to *stress* the server by putting it under a very high request load. When the server is almost or totally jammed with requests, the time gaps in response are monitored and we can determine the systems behavior on the occurring situation. Here, we want to evaluate, if the implemented mechanisms are going to resolve the client's request congestion in the satisfying manner. In the evaluation scenario with a Java client model, we are trying to simulate the build up of flash crowds and to monitor the system's behavior in the situation, which represents the normal working environment and the most common practice.

The Java client model consists of the HTTP protocol implementation for establishing the HTTP connection with the server, time scheduler and the redirection following mechanism (See Code Listing 6.1). The HTTP connection is initiated from virtual clients and sent to the target server using the URL address. The frequency of sending the URL Requests is controlled by the time scheduler, which is implemented in Java.

The time scheduler is a simple Java implementation of the timer, that is used for scheduling of the client requests. The time scheduler can be set up with parameters, which denote the time period between two client requests. If we use a smaller time gap between two client requests, the request rate will be higher, and vice versa. For achieving a high time scheduling precisions, we used a nanosecond timer for client request scheduling. Once the HTTP response is received from the server side, the information about the request completion can be extracted and used for statistics. In Java there is a flag called *setFollowRedirects*, which can be set when using the HTTP protocol implementation for establishing the HTTP connection. This flag is implemented in the *URLConnection* Java class. In this evaluation scenario, several machines on the academic testbed, called PlanetLab [36, 37], were used to start the Java Client Model and access the central server positioned in the IAM institute, University of Bern. The surrogate server was also positioned in the IAM institute. The Java client model simulated many instances of the client requests. The number of requests per time unit was parametrized and was changed during the evaluation period. In this scenario, the surrogate server's role was not important. The main focus here was to measure the ratio between the number of redirected client requests and the total client requests send to the origin server. Figure 6.1 illustrates the RER test scenario with a Java Client Model.

Code listing 6.1 represents the Java client model implementation and demonstrates the Java class, which is responsible for establishing the HTTP HTTP connection with the

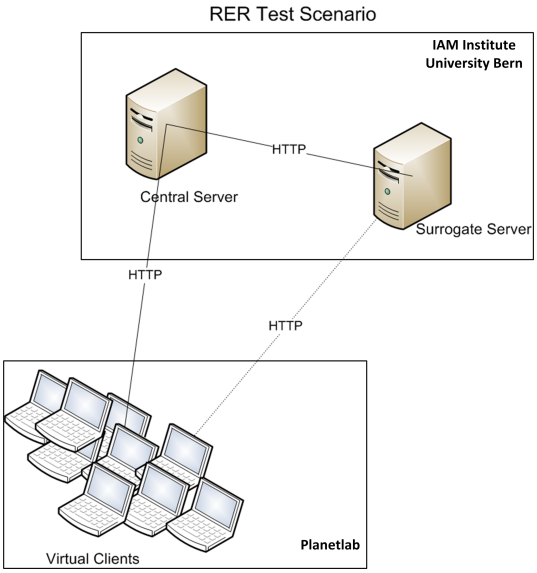

Figure 6.1: RER Test Scenario with Java Client Model

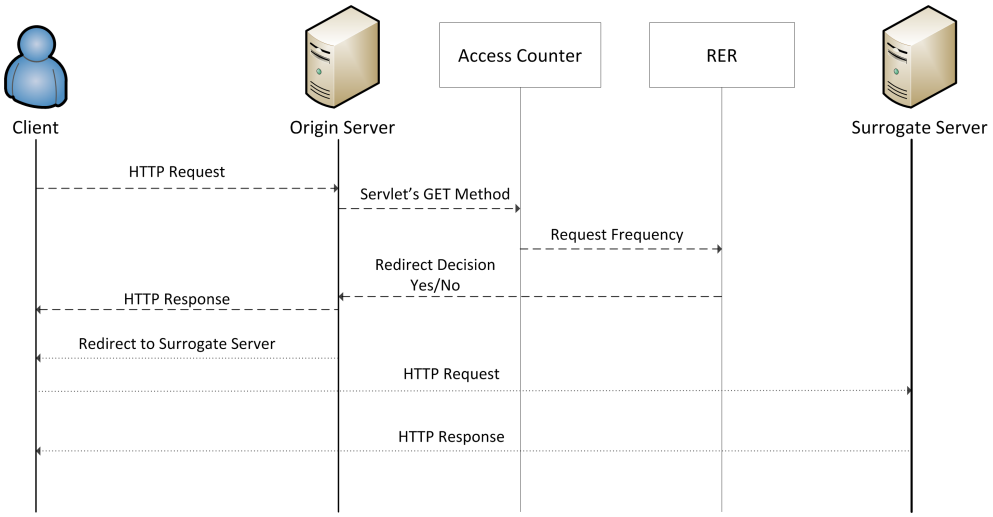

Figure 6.2: Client Communication Scenario with Origin and Surrogate Server

server using the URL address. The table also shows the *setFollowRedirects* flag used to count the number of redirects on the HTTP connection, the so called HTTP response code, taken as parameter from the HTTP connection class, and the if/else statement, which is used for distinguishing of the HTTP response code. The HTTP response code **302** indicates that a client request has been redirected, whereas the HTTP response code **200** stands for the served client requests. With the if/else statement, we can add counters, which collect evaluation results for different request rates. The typical content size on the origin server amounted to a few Megabytes. Figure 6.2 shows the communication scenario between the client and origin server. The origin server has two possibilities, either to response to the HTTP request or to send the redirect code to client, which will be automatically redirected to the surrogate server.

Listing 6.1: Java Code for Following the Request Redirections on Central Server

```
urlConnection = (HttpURLConnection)url.openConnection();
urlConnection.setFollowRedirects(false);
int responceCode = urlConnection.getResponseCode();
 if(responceCode == 302)
 {
         redirectCount++;
         System.out.println("redirectCount :" + redirectCount);
 }else if (responceCode == 200)
 {
         deliveredCount++;
         System.out.println("deliveredCount :" + deliveredCount);
 }
```

In our evaluation scenario, we measured the percentage of redirected requests in correlation with the number of requests per unit time in seconds. The evaluation was initiated with 50 client requests per second with an increase of 10 client requests. During the evaluation period, this increase was added repeatedly to the existing number of client requests per unit time until 300 requests per second were reached. To avoid the measurement errors, the evaluation with the Java client model was repeated 50 times. According to the statistical calculations, all measurements greater than 30 tends to assume a normal distribution. The evaluation results are illustrated in the Figure 6.3. The graph in the Figure 6.3 shows the confidence interval of +/- one standard deviation higher/lower than the mean value. At the normal distribution, the confidence interval of +/- one standard deviation compared to the mean value includes about 68% of statistical measurement data. The graph also shows that the percentage of redirects equals 0 in the first few steps (from 50 to 90 requests per sec.). This illustrates the minimum threshold applied in the RER algorithm. In the range from 50 to 90 seconds, the server's role is pure content provider, which fulfills every client request. From 100 to 300 requests per second, the server's role changes to partial load balancer and content provider and the ratio between the number of redirects and the number of total requests is rising. Here we can see clearly the very similar function line behavior as in the Figure 3.1, although the results are not equivalent to the real RED graph, since there is a timing gap in the Java scheduler. Obviously the Java performance bottlenecks,

like garbage collector, are influencing the RED behavior. Nevertheless, this result gives us a valuable feedback about the RER implementation in Gezora and its responsiveness, which permitted further Gezora implementation- and evaluation steps.

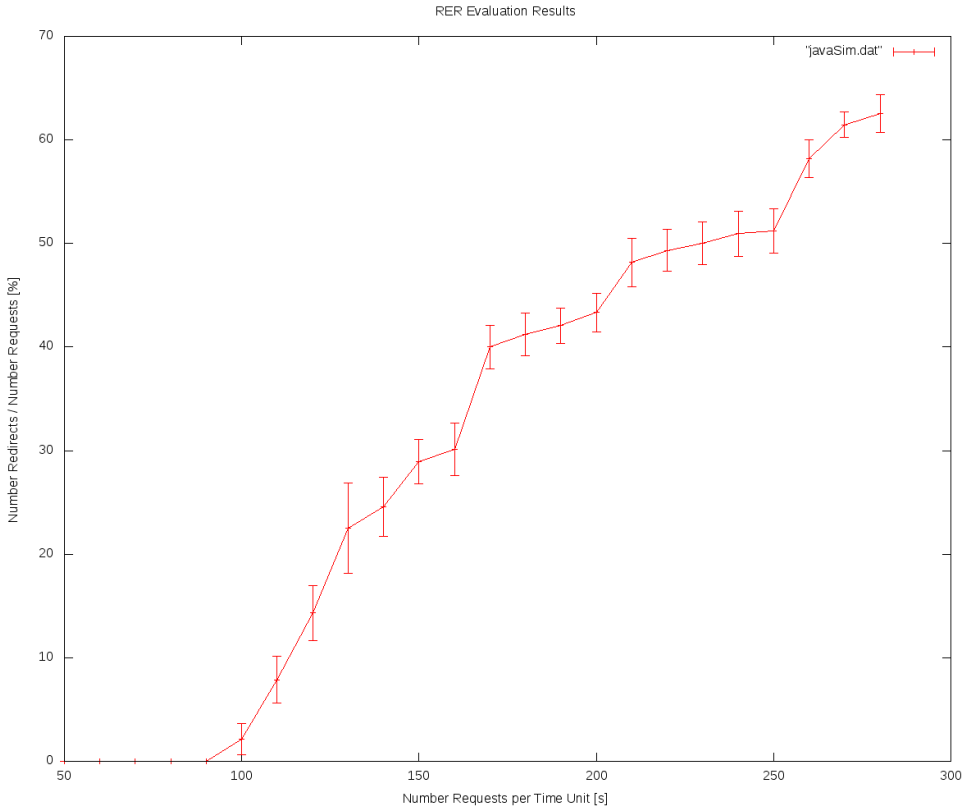

Figure 6.3: RER Evaluation Results

The Figures B.1, B.2 and B.3 give the detailed overview about the Java client evaluation model, presented as the sequence diagrams.

6.4 Gezora Performance Evaluation

A web server stress tool emulates the client activity using a realistic client simulation. It generates the so-called virtual clients and measures the various server performance parameters. The web server stress tool is designed to determine the system's behavior in case of congestion. In other words, this test is designed to test the system dynamically, which emphasizes the importance of the test. Its task is to simulate the most common situations, which the system will encounter in real life exploitations, most notably the build up of flash crowds. The measured parameters can give a meaningful feedback about the server's performance bottlenecks. The main functionality of web stress tests reflects the

fact that each virtual client with his emulated functions acts as a real user. Still, this kind of simulation has certain limits, regarding the maximum number of the emulated simultaneous client requests. Using a web stress tool, a maximum of 10'000 simultaneous client requests can be simulated without suffering from the limitations of the single computer test environment.

One web server test can emulate the load of thousands of virtual users experienced by a web site when the corresponding number of real users accesses it. User profiles describe the behavior of virtual users. This allows us to emulate a real workload on the server. One user profile consists of a specific number of virtual clients who process the HTTP transactions during the specified time period. For each user profile, the web page URL addresses are specified. We applied a test scenario, where a load increase is made after specified time intervals. The test duration and the maximum number of virtual clients per time unit for each user profile can also be determined. At the end of the tests, the summary reports and graphs are generated. With the help of the summary reports and graphs, we obtained a specific estimation of Gezora's performance during the high demand time periods. More specifically, with help of the web stress tool, we are able to answer the following questions:

1. Is the web server prepared for the expected traffic?
2. Is the web server prepared for increasing visitors over time?
3. Can the web server survive sudden traffic spikes?
4. How many users can the web server handle before error messages and server timeouts occur?
5. How much time does it take for a client to receive a page after clicking on a link?
6. Are the scripts and databases optimized to run as quickly as possible and do they interact with each other correctly under heavy web server loads?
7. Is the web server bandwidth and hardware sufficient?

6.4.1 Web Server Performance Evaluation Types

The most prominent test types in the web server stress application are:

1. Performance Tests
2. Stress Tests
3. Ramp Tests

6.4.1.1 Performance Tests

Performance tests are applied to the web server to determine the web server performance under increased web traffic. During the test period, several simultaneous requests are sent

to one URL and the average client request time is recorded. Usually, performance tests run without requesting web page content like images. They are concentrated on the script and code testing itself. This simple core tests are aimed at testing the basic functioning of the system. If a basic functioning of the system is flawed, there is not much sense in testing the system further with more complicated tools.

6.4.1.2 Stress Tests

Stress tests are simulated denial of service attacks. With stress tests, the real situations, like sudden traffic spikes, can be easily generated and sent to the web server. The purpose of a stress test is to estimate the maximum load at which the web server can handle and also to learn the web server's traffic thresholds and determine the response behavior after exceeding the thresholds. This kind of test is intended to test the system performance statically. The maximum number of users is determined and the behavior of the system is monitored. In this scenario administrators are trying to test the server by putting it into the border line situation. When the server is almost or totally jammed with requests, it is monitored in order to determine how the system will handle the situation. This is the situation where the system is already congested and administrators need to see is it going to resolve the situation in the satisfying manner.

6.4.1.3 Ramp Tests

Ramp Tests are a subcategory of stress tests. The main difference to stress tests is that ramp tests increase the number of users during the test time dynamically, from one to thousands of users, compared to the stress tests, where one fix amount of requests is generated to trigger the server overloading. Ramp tests are more flexible than standard stress tests, since they are able to test various load rates on the server. With the help of ramp tests, we can determine what the maximum load of one server can handle while providing optimal access time to web content. The acceptable web server response time should be less than 10 seconds. When users experience poor performance on a web site, they are dissatisfied, and may leave never to return again. In this scenario, the test is designed to determine the system behavior in the case of traffic congestion.

Perhaps this is the most important kind of test, because its task is to simulate the most common situations, which the system will encounter in real life exploitations the build up of flash crowds. System and network administrators are trying to simulate the build up of flash crowds and to monitor the system's behavior in the situation which represent the normal working environment and the most common case in practice. Because of this, a ramp test is probably the most interesting kind of test which also generates the highest data volume, the most complex data and is the closest to real life exploitation test results.

6.4.2 Evaluation Scenarios using Web Server Stress Tests

For Gezora performance testing, we applied RAMP Tests, described in the Section 6.4.1.3. Our evaluation scenario consisted of a variable number of emulated client requests ranging from minimum 1 request per second to a maximum of 1500 requests per second. A simulated request number is increased during the test period adding the variable of request number every 30 seconds. The web stress tool was running on two different servers, positioned in the IAM institute at University of Bern. This doubled the emulated request rate up to 3000 simultaneous requests per second. The origin server was also positioned in the IAM institute and connected with a surrogate server. Figure 6.4 illustrates the Gezora performance evaluation scenario using web server stress tests.

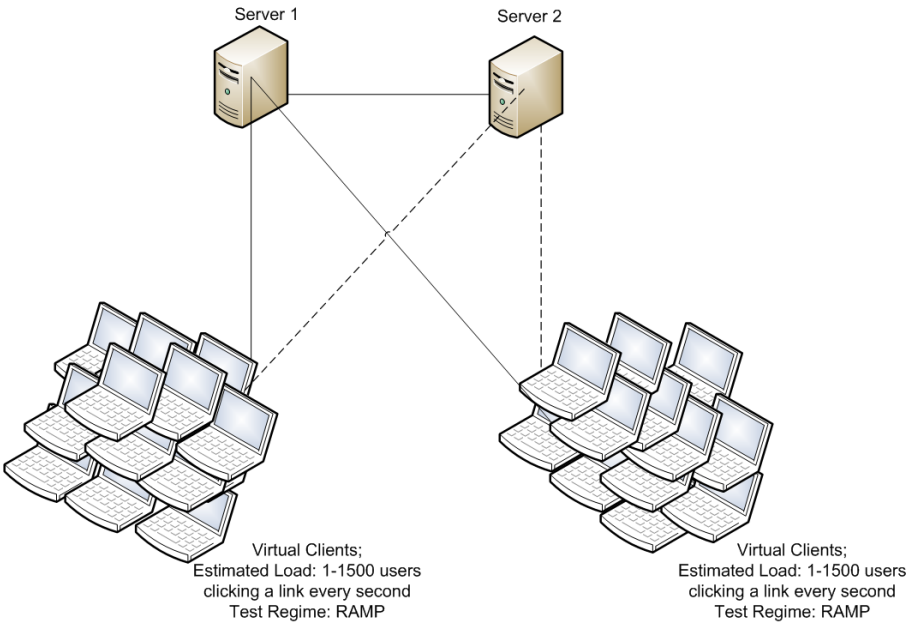

Figure 6.4: Gezora Web Server Stress Tests Scenario

For evaluation purposes, we implemented a small web page with an estimated content of ca. 26 MB. The structure of the web site had a very simple design and contained only static content (text and pictures). With referencing of the main URL address of the web page (Main.jsp), the web content of 26 MB was transferred to the client and showed on the web page. The data transfer was accomplished using HTTP communication protocol previously described in the Section 5.3.5.

6.4.3 Evaluation Results with Web Server Stress Tests

In order to avoid potential measurements errors, the same web stress evaluation was repeated 100 times, using the test scenario described in the Section 6.4.2.

Figure 6.5 illustrates Gezora evaluation results and shows the average request time in correlation with the request number per time unit (second), after applying 100 RAMP tests to the original web server without Gezora CDN and to the Gezora CDN, which consisted of one original and one surrogate server. The graph in the Figure 6.5 shows a clear difference in the average request time when Gezora is utilized and when only one origin server is utilized. In the range from 200 requests/sec to around 1400 requests/sec, the average request time of one origin server without Gezora increases drastically as compared to the average request time measurements of a Gezora CDN with one surrogate, which is increases slightly and remains below 100 ms. In the range from 1300 requests/sec to 1500 requests/sec the request time of one origin server without Gezora is increasing even faster and achieves a maximum request time of 600 ms, whereas the Gezora CDN request time reaches the maximum value of 180 ms. This is a very important evaluation result, which shows the difference in the request time between one origin server and Gezora CDN with minimal resources, namely only one surrogate. The difference in the client's request time during the peak request rate was about 30% smaller using a Gezora CDN. This proves that our request frequency monitoring mechanism, based on EWMA and RER redirection mechanism reacts fast to the request frequency changes, which can use the volunteer server's resource for the mitigation of the request congestion. This mechanism is very flexible, and it can be applied not only for large server scaling, but also for moderate CPU scaling using many CPU cores. The resources from every participating volunteer server should be used from the beginning, before a real congestion occurs.

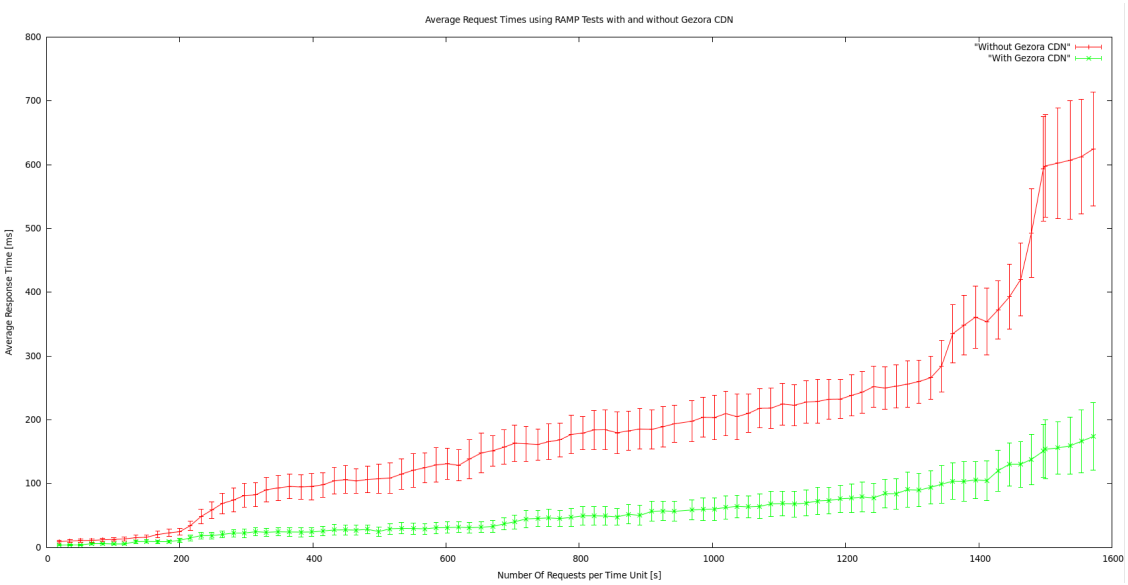

Figure 6.5: Average Request Time [ms] in Correlation with the Request Number per Time Unit [s]

Further Gezora evaluation results, presented in Figures 6.6, 6.7, 6.8, 6.9 and 6.10 illustrate the percentage of clients who waited for the some web content during the specific time period.

Most importantly, the correlation between the waiting time during the request and the number of clients is presented. This graph shows the percentage of users (compared to the total number of user requests), who waited a certain time period. Figures 6.6, 6.7, 6.8, 6.9 and 6.10 clearly state the difference between the results with and without Gezora application.

For example, we can notice on Figure 6.6 the difference between two lines (with and without Gezora). A green line denotes the percentage of users who waited less than 20 ms with Gezora, and the red line denotes the percentage of users who waited less than 20 ms without Gezora. Here we can see that the green line is holding the 100% of users up to around 600 requests per second. After this value, it starts to fall down. This indicated the changing of the request time in correlation with the increase of request number per unit time (s). On the other side, the red line behavior differs from the first one. It starts to fall already at 200 requests per second, and is falling faster than the first one.

Other figures show exactly the same behavior of lines for the different waiting times. If we compare the results in Figure 6.10, a clear difference in trends is visible. Namely, the percentage of users who waited less than 200 ms in the range of a high user request rate was much higher than in the system with one origin server, without Gezora CDN. The green line is holding 100% of users all time, whereas the red line starts to fall from 800 request/s on.

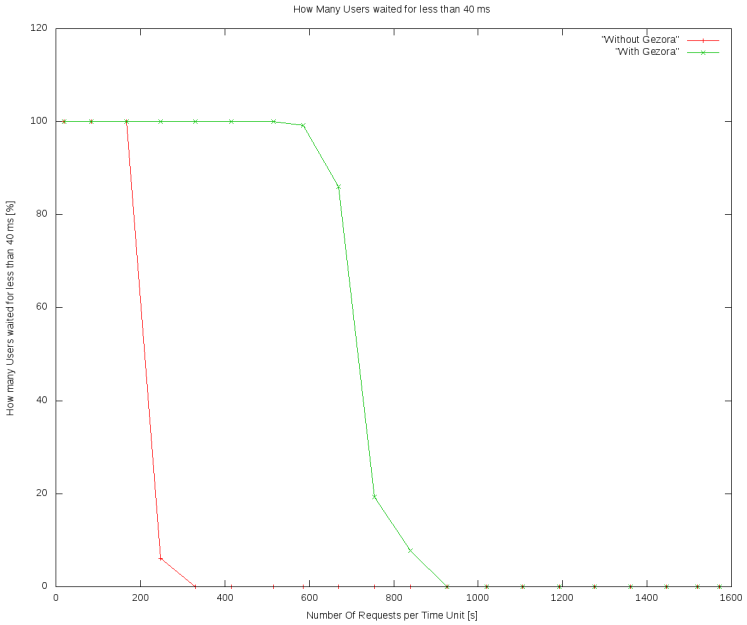

Figure 6.6: Percentage of Users with a Request Time less than 20 ms

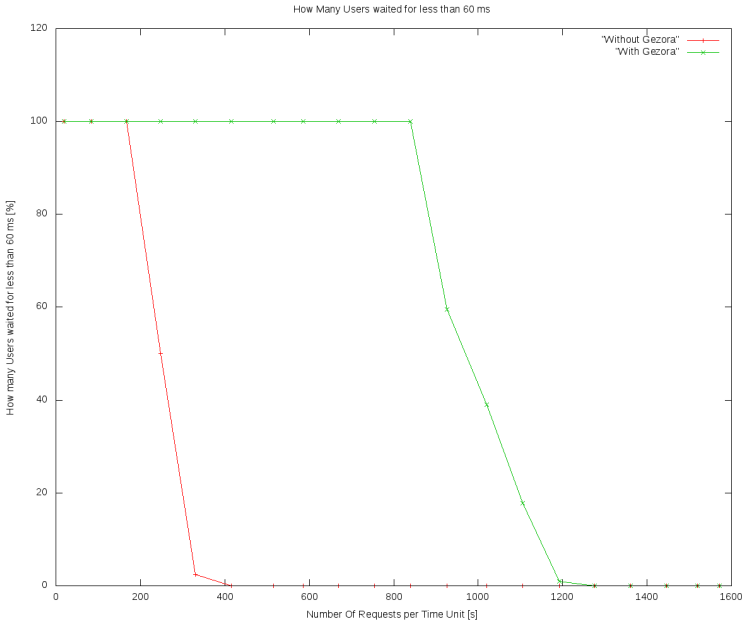

Figure 6.7: Percentage of Users with a Request Time less than 60 ms

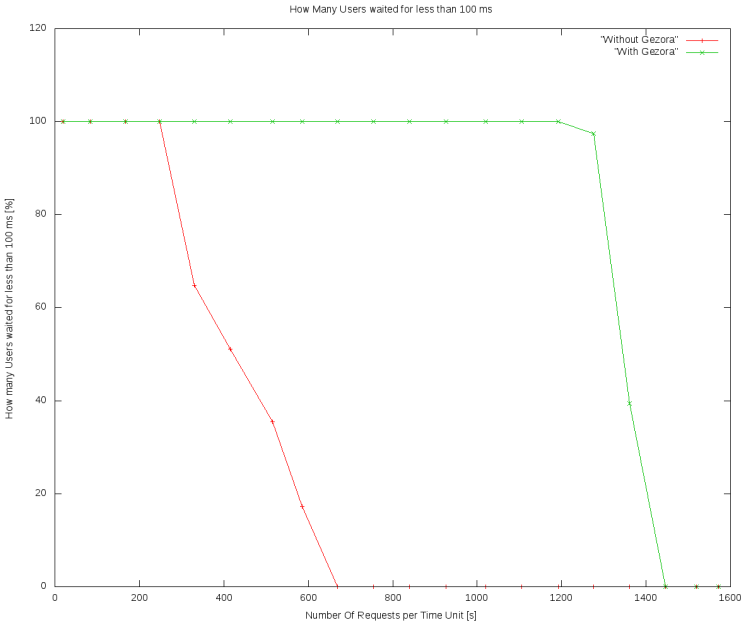

Figure 6.8: Percentage of Users with a Request Time less than 100 ms

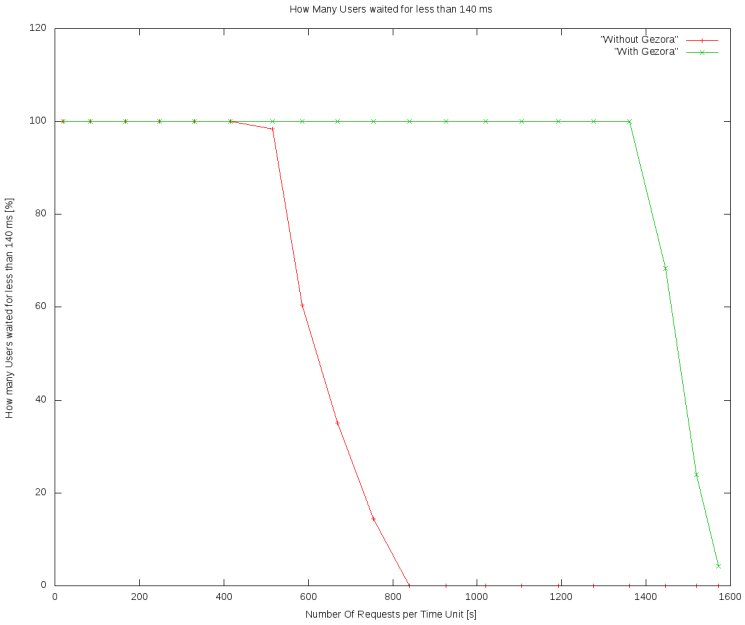

Figure 6.9: Percentage of Users with a Request Time less than 140 ms

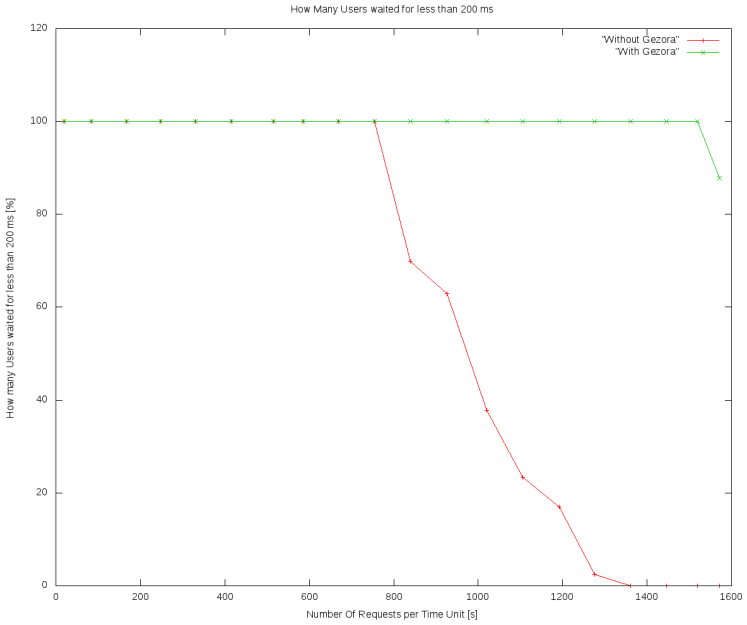

Figure 6.10: Percentage of Users with a Request Time less than 200 ms

The described evaluation results give a clear statement and an important feedback about Gezora's scalability during the high demand period.

Chapter 7

Conclusions and Outlook

The main goal of this master thesis was to present the main aspects of Gezora's design and functionality as well as the advantages over other systems. The main efforts as well as the complex design were developed in order to accomplish only one requirement, which is serving user's requests as fast and as easy as possible. This aspect is important not just for users, which demand a high system and network performance, but for the content providers as well.

We implemented a new kind of CDN prototype, which consists of the request frequency monitoring system (EWMA Filter), random early redirection (RER) and request routing mechanisms, such as URL rewriting and HTTP redirection. The functional prototype was implemented in Java programming language using J2EE and Java servlets. Gezora's essential task was to decrease the latency and increase the speed of serving the user's request, or to deliver the content to the end users as quickly and easily as possible.

The Gezora evaluation gave a very important feedback about performance regarding the adapting to the fast request number changes (flash crowds). We applied a simulation of a real-life scenario from the end user side and we applied a real-life server infrastructure for our evaluations. As, mentioned earlier, there is no better way to test any kind of system as in a real life testing environment. The downside of real-life testing is that it is not always possible to test all border line cases and parameters, because those cases occur very rarely, the evaluation costs for this kind of evaluation could be high, or time to perform the testing could be very long. In such situations, the mathematical models and computer simulations, which we have just mentioned, can give satisfying results, with a high accuracy. Still, we must have in mind that the computer simulations are not 100% reliable.

Considering other evaluation methods, the best solution would be to perform a mixed testing, by testing the system off-line first and than allowing the users to test it on-line as a beta version in more or less controlled or pre-determined conditions. This mixed approach of modern mathematical, statistical and software tools and models with real life testing has

given the best and the most satisfying results so far, by the lowest costs possible and the highest possible accuracy. Unfortunately, nothing is ever 100% error proof, but a mixed approach experience has showed that it leaves the least work on system maintenance after the system has been tested by this approach. Some other methods may be used for testing and simulating the working environment prior to the actual system deployment. We leave it to every researcher to choose their most preferred testing method, but one thing is clear to all, and that is the importance of prior system testing. The importance of prior testing is among the most important steps in system design and deployment as much as real life smooth functioning and stability is. The latency perceived by the client is in the direct correlation with the quality and one of system's main quality factors (stability, decreased latency, good experience during exploitation, system proactivity etc.).

According to the evaluation results, Gezora accomplished a roughly 30% better request time across all access rates using only one surrogate server compared to the system with one origin server. This indicates that a RER algorithm can be applied to the future CDNs for mitigation of the fast changing request rates. All tests and graphs indicate that Gezora is able to employ the modular design and existing technology in a new way and deliver much needed performance by reasonable costs. Today in the information technology era, when the user's wish is the law, the primary request for any system is to satisfy user requests in a minimum amount of time as reliably and as cheaply as possible, and Gezora is surely up to the task.

During the Gezora prototype implementation and evaluation process we concluded that there is much space for improvements in the Gezora design. For the future work we are planning to implement the feedback control loop described in 3.6, which would additionally include a weight-fairness (α parameter) in the EWMA frequency filter. In our future work, we would also investigate the impact of the additional feedback control for weight adaptation on the evaluation results. Further work would focus more on finding the scaling factor, which correlates to the number of surrogate servers for different load rates and to derive the mathematical expression for it.

Bibliography

[1] Wikipedia, "Gezora kaiju." [Online]. Available: http://godzilla.wikia.com/wiki/Gezora

[2] G. F. Franklin, D. J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," 1993.

[4] D. Carra, "Content delivery in overlay networks: a stochastic graph," in *Processes Perspective, in Proc. IEEE GLOBECOM 2006, Nov. 27*, 2006.

[5] Wikipedia, "Youtube." [Online]. Available: http://de.wikipedia.org/wiki/YouTube

[6] "Facebook." [Online]. Available: http://de.wikipedia.org/wiki/Facebook

[7] A. mukaddim Khan Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," 2008.

[8] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Communications of the ACM*, vol. 49, pp. 101–106, 2006.

[9] S. Androutsellis-theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, pp. 335–371, 2004.

[10] "Scalr: An on-demand scaling system for amazon cloud services." [Online]. Available: https://www.scalr.net/

[11] "Akamai: Web application acceleration and performance." [Online]. Available: http://www.akamai.com/

[12] Wikipedia, "Transmission control protocol." [Online]. Available: http://de.wikipedia.org/wiki/Transmission_Control_Protocol/Internet_Protocol

[13] C. Technologies, "Winpcap: Windows packet capturing library." [Online]. Available: http://www.winpcap.org/

[14] B. Mobasher, R. Cooley, and J. Srivastava, "Creating adaptive web sites through usage-based clustering of urls," in *In IEEE Knowledge and Data Engineering Workshop (KDEX'99*, 1999.

[15] Y. C. Uc, Y. Chen, L. Qiu, W. Chen, L. Nguyen, and Y. H. Katz, "Clustering web content for efficient replication," in *Proceeding of the 10th IEEE International Conference on Network Protocols*. Society Press, pp. 165–174.

[16] Y. Chen, L. Qiu, W. Chen, L. Nguyen, R. H. Katz, and Y. H. Katz, "Efficient and adaptive web replication using content clustering," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 979–994, 2003.

[17] Y. M. Liu and I. Traore, "Systematic security analysis for service-oriented software architectures," in *Proceedings of the IEEE International Conference on e-Business Engineering*, ser. ICEBE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 612–621. [Online]. Available: http://dx.doi.org/10.1109/ICEBE.2007.137

[18] N. K. Nagwani, "Clustering based url normalization technique for web mining," *Advances in Computer Engineering, International Conference on*, vol. 0, pp. 349–351, 2010.

[19] "Url normalisation." [Online]. Available: http://en.wikipedia.org/wiki/URL_normalization

[20] A. Ailamaki, C. Garrod, C. Olston, B. Maggs, A. Manjhi, G. Inc, A. Tomasic, and T. Mowry, "Akamai technologies."

[21] "Mirror image: Digital media delivery." [Online]. Available: http://www.mirror-image.com/

[22] M. J. Freedman, E. Freudenthal, and D. M. Eres, "Democratizing content publication with coral," in *In NSDI*, 2004.

[23] Wikipedia, "Round robin algorithm." [Online]. Available: http://en.wikipedia.org/wiki/Round-robin

[24] C. C. Group, "Large hadron collider." [Online]. Available: http://public.web.cern.ch/public/en/lhc/lhc-en.html

[25] "European virtual observatory." [Online]. Available: http://www.euro-vo.org/pub/

[26] "Biogrid: Database of protein and genetic interactions." [Online]. Available: http://thebiogrid.org/

[27] "Oracle exadata storage servers." [Online]. Available: http://www.oracle.com/technetwork/database/exadata/dbmachine-x2-8-datasheet-173705.pdf?ssSourceSiteId=ocomen

[28] E. W. Biersack, P. Rodriguez, and P. Felber, "Performance analysis of peer-to-peer networks for file distribution," in *In Proc. Fifth International Workshop on Quality of Future Internet Services*, 2004.

[29] Wikipedia, "Bittorrent p2p." [Online]. Available: http://de.wikipedia.org/wiki/BitTorrent

[30] "Xerox palo alto research center." [Online]. Available: http://www.parc.com/

[31] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson, "Reliability and security in the codeen content distribution network," in *In USENIX Annual Technical Conference, General Track (2004*, pp. 171–184.

[32] T. T. Proactive, P. Enterprise, and M. M. The, "Planetlab and its applicability," 2004.

[33] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating system support for planetary-scale network services," 2004, pp. 253–266.

[34] M. Freedman and D. Mazieres, "Sloppy hashing and self-organizing clusters," in *In IPTPS*, 2003, pp. 45–55.

[35] U. d. G. Matteo Dell'Amico, Dipartimento di Informatica e Scienze dell'Informazione, "Highly-clustered networks with preferential attachment to close nodes," 2006.

[36] N. Spring, L. Peterson, A. Bavier, and V. Pai, "Using planetlab for network research: myths, realities, and best practices," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 17–24, 2006.

[37] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences building planetlab," in *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 351–366.

[38] Wikipedia, "Network file system." [Online]. Available: http://de.wikipedia.org/wiki/Network_File_System

[39] S. Annapureddy and M. J. Freedman, "Shark: Scaling file servers via cooperative caching," in *In Proc NSDI*, 2005. [Online]. Available: http://www.scs.cs.nyu.edu/shark/

[40] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Cambridge, MA, USA, Tech. Rep., 1979.

[41] J. Friedman and E. Garrison, "Building a hierarchical content distribution network with unreliable nodes." [Online]. Available: http://www.scs.cs.nyu.edu/shark/

[42] S. L. Garfinkel, I. Voip, and S. Security, "Skype security overview, rev 1.6- 1/26/05 by."

[43] P. Goering and G. Heijenk, "Service discovery using bloom filters," in *In: Proceedings Twelfth annual conference of the Advanced School for Computing and Imaging*, pp. 14–16.

[44] Y. Matsumoto, H. Hazeyama, and Y. Kadobayashi, "Adaptive bloom filter: A space-efficient counting algorithm for unpredictable network traffic," *IEICE - Trans. Inf. Syst.*, vol. E91-D, pp. 1292–1299, May 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1522798.1522838

[45] A. A. J. Jehoshua Bruck, Jie Gaoy, "Adaptive bloom filter."

[46] "Servlet api documentation." [Online]. Available: http://tomcat.apache.org/tomcat-5.5-doc/servletapi/index.html

[47] "Apache tomcat: An open source software implementation of the server." [Online]. Available: http://tomcat.apache.org/

[48] "Jboss enterprise: A community-driven open source middleware." [Online]. Available: http://www.jboss.org/

[49] M. Corporation, "Ms windows server 2008 r2 os." [Online]. Available: http://www.microsoft.com/windowsserver2008/en/us/default.aspx

[50] "Red hat enterprise linux server os." [Online]. Available: http://www.redhat.com/rhel/server/

[51] P. Pradhan, R. Tewari, S. Sahu, A. Ch, and P. Shenoy, "An observation-based approach towards self-managing web servers," in *In Proceedings of the Tenth International Workshop on Quality of Service (IWQoS 2002*, 2002.

[52] R. Zhang, S. Parekh, Y. Diao, M. Surendra, T. Abdelzaher, J. Stankovic, R. Zhang, Y. Diao, T. Abdelzaher, S. Parekh, M. Surendra, and J. Stankovic, "Control of weighted fair queueing: Modeling, implementation, and experiences," 2004.

[53] "Keynote systems: The mobile and internet performance authority." [Online]. Available: http://www.keynote.com/

[54] "Giga information group." [Online]. Available: http://www.business.com/directory/computers_and_software/reference/giga_information_group/profile/

[55] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang, "Idmaps: A global internet host distance estimation service," in *In Proceedings of IEEE INFOCOM*, 2000, pp. 210–217.

[56] "Omnet++ network simulation framework." [Online]. Available: http://www.omnetpp.org/

[57] "Inet framework: An open-source communication networks simulation package for the omnet++ simulation environment." [Online]. Available: http://inet.omnetpp.org/

[58] I. Paessler, "Paessler: Network monitoring software, network performance management." [Online]. Available: http://www.paessler.com/webstress/

Appendix A

UML and Sequence Diagrams: Gezora Prototype

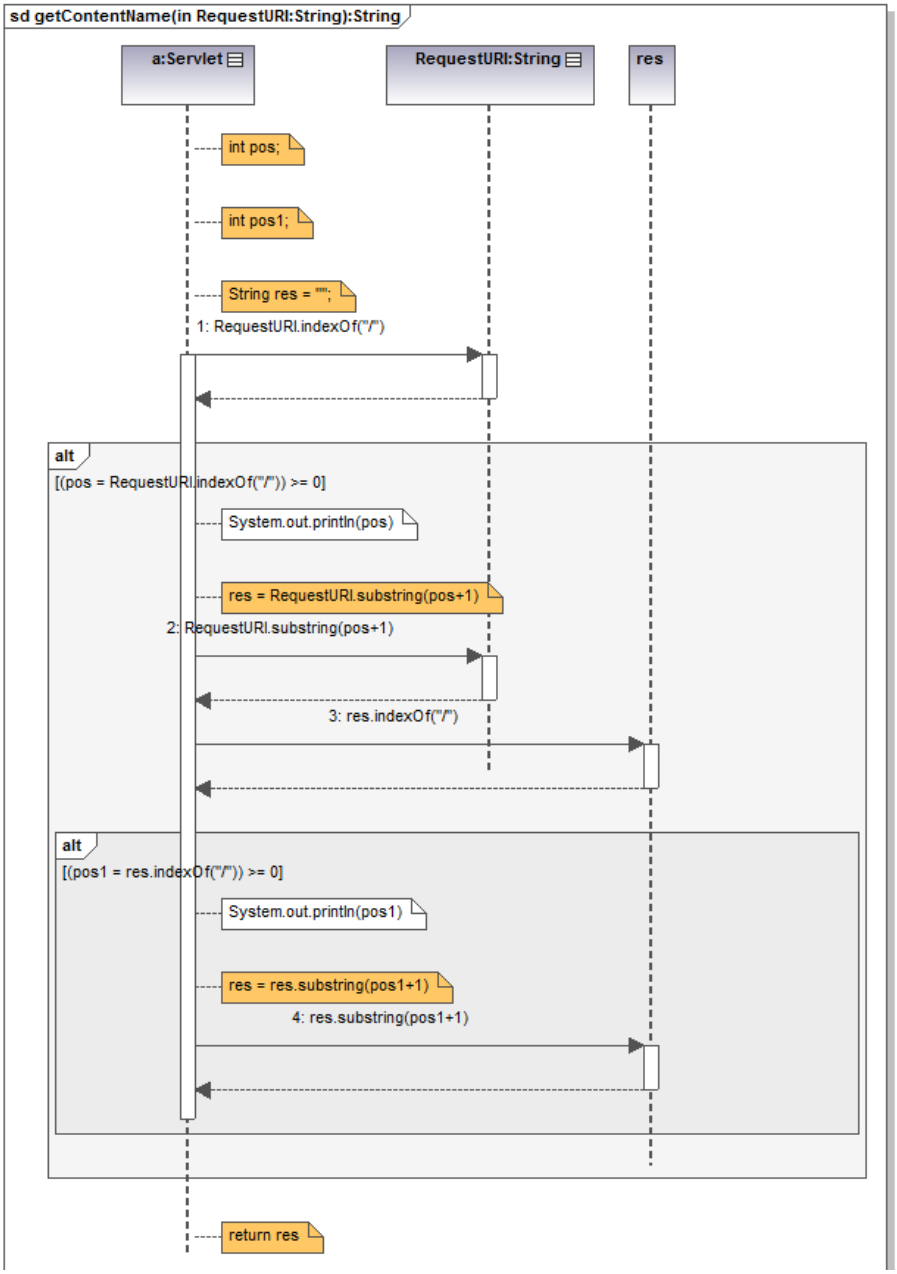

Figure A.1: Getting Content Name

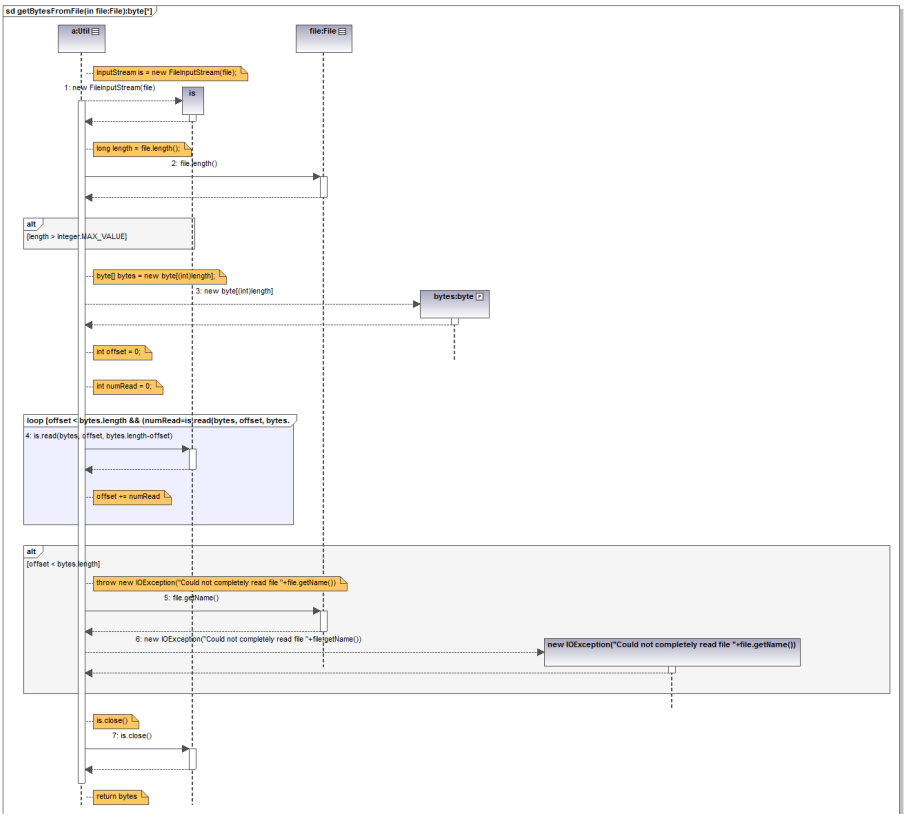

Figure A.2: Java Stream Implementation for Content Transfer

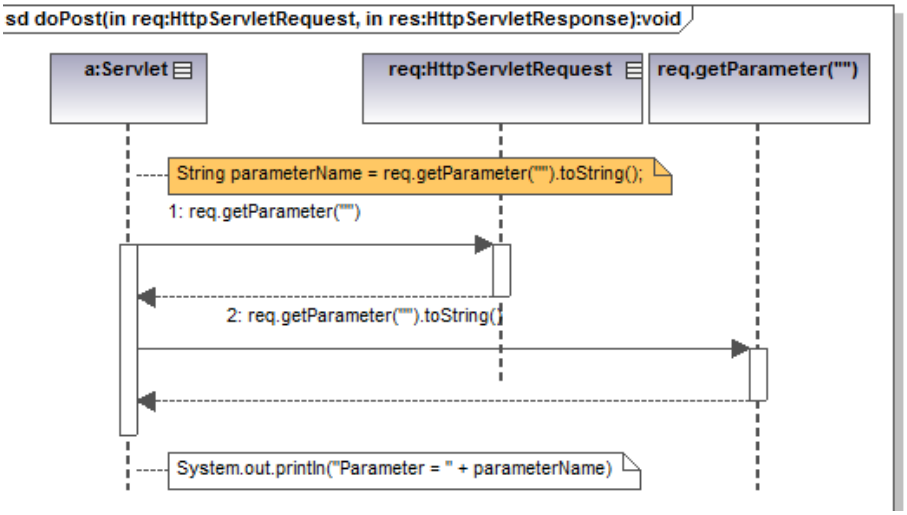

Figure A.3: HTTP doPost Method

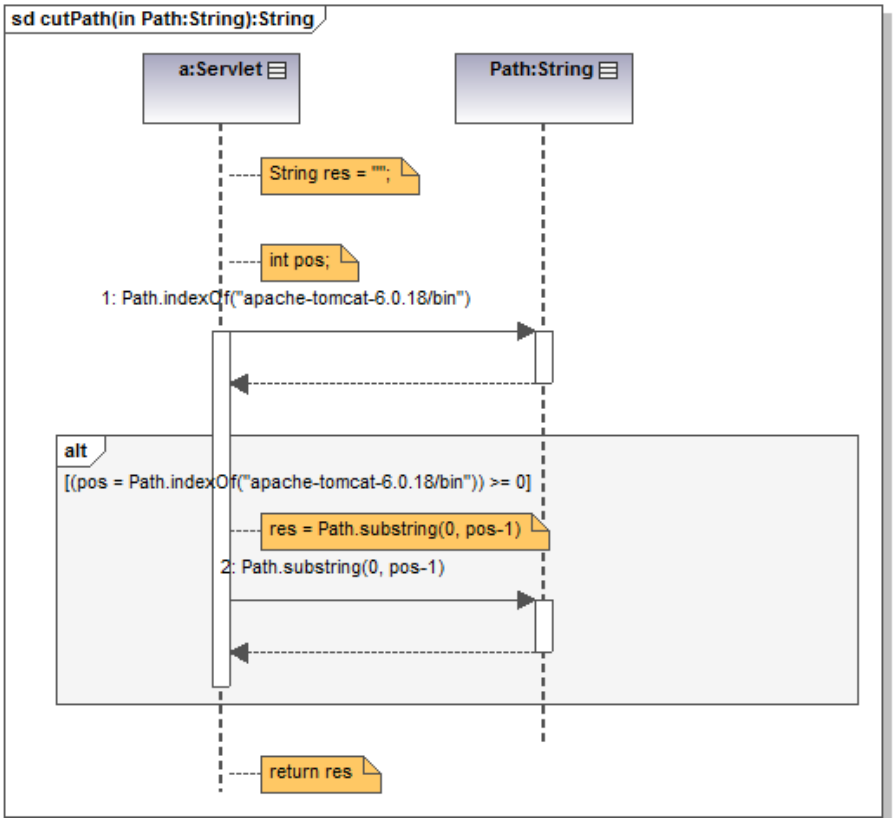

Figure A.4: Java Method for Cutting the URL Path

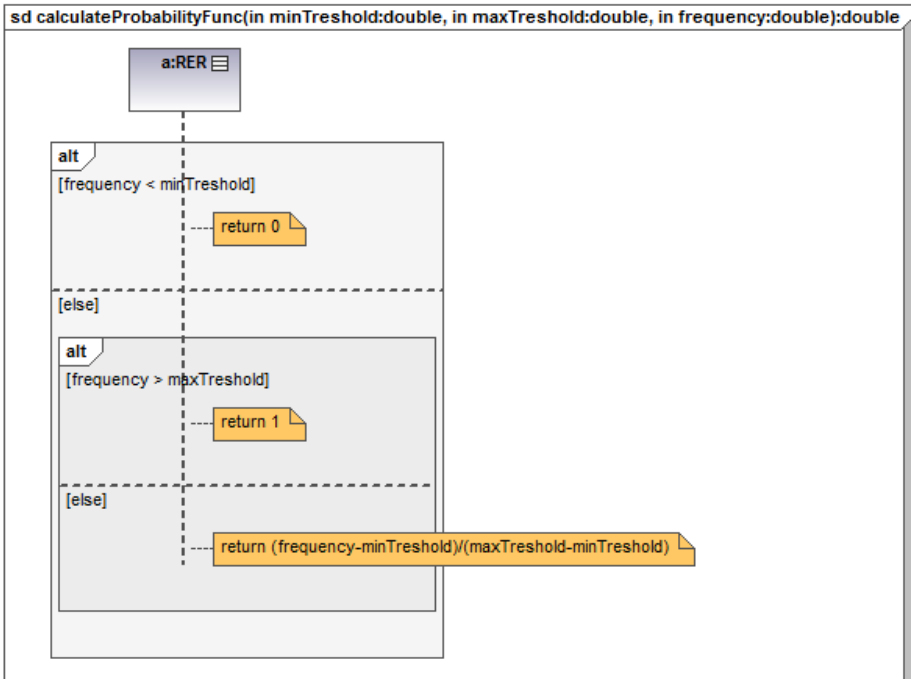

Figure A.5: Probability Calculation Implemented in Java

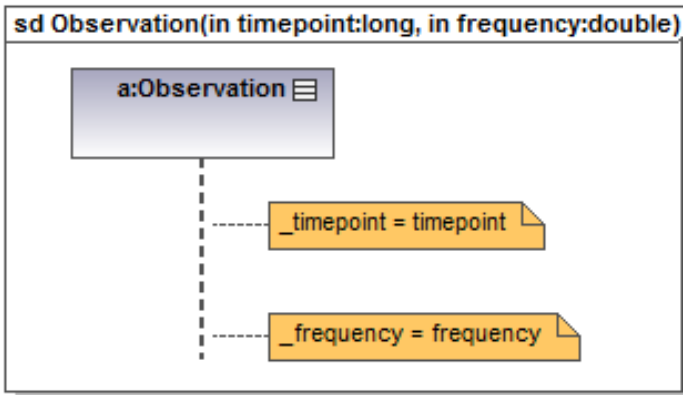

Figure A.6: Java Object with two Parameters Holding the Time and Frequency of Client Requests

Appendix B

UML and Sequence Diagrams: Java Client Model

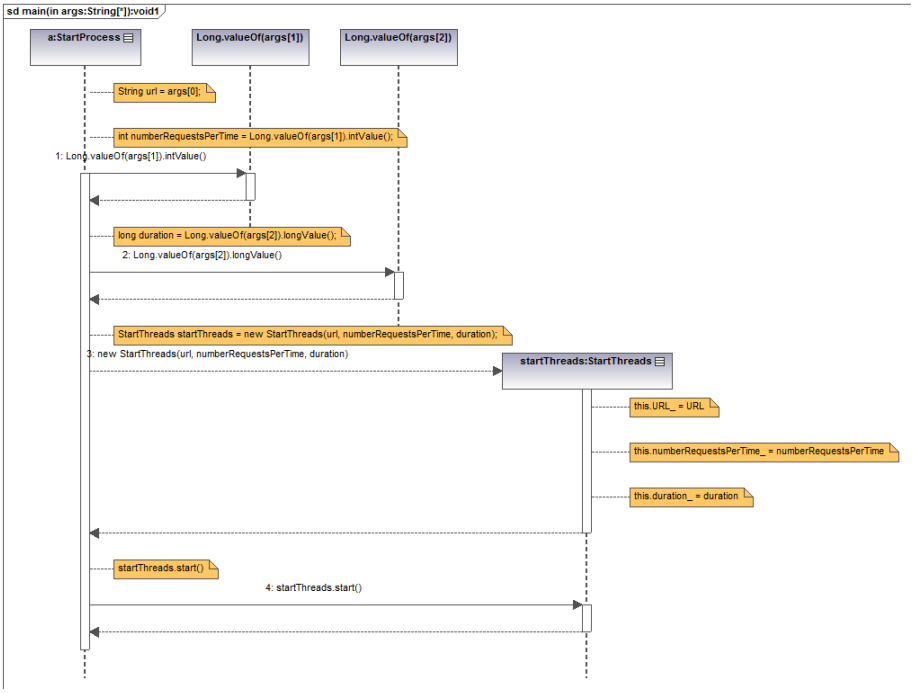

Figure B.1: The Core of the Java Client Model

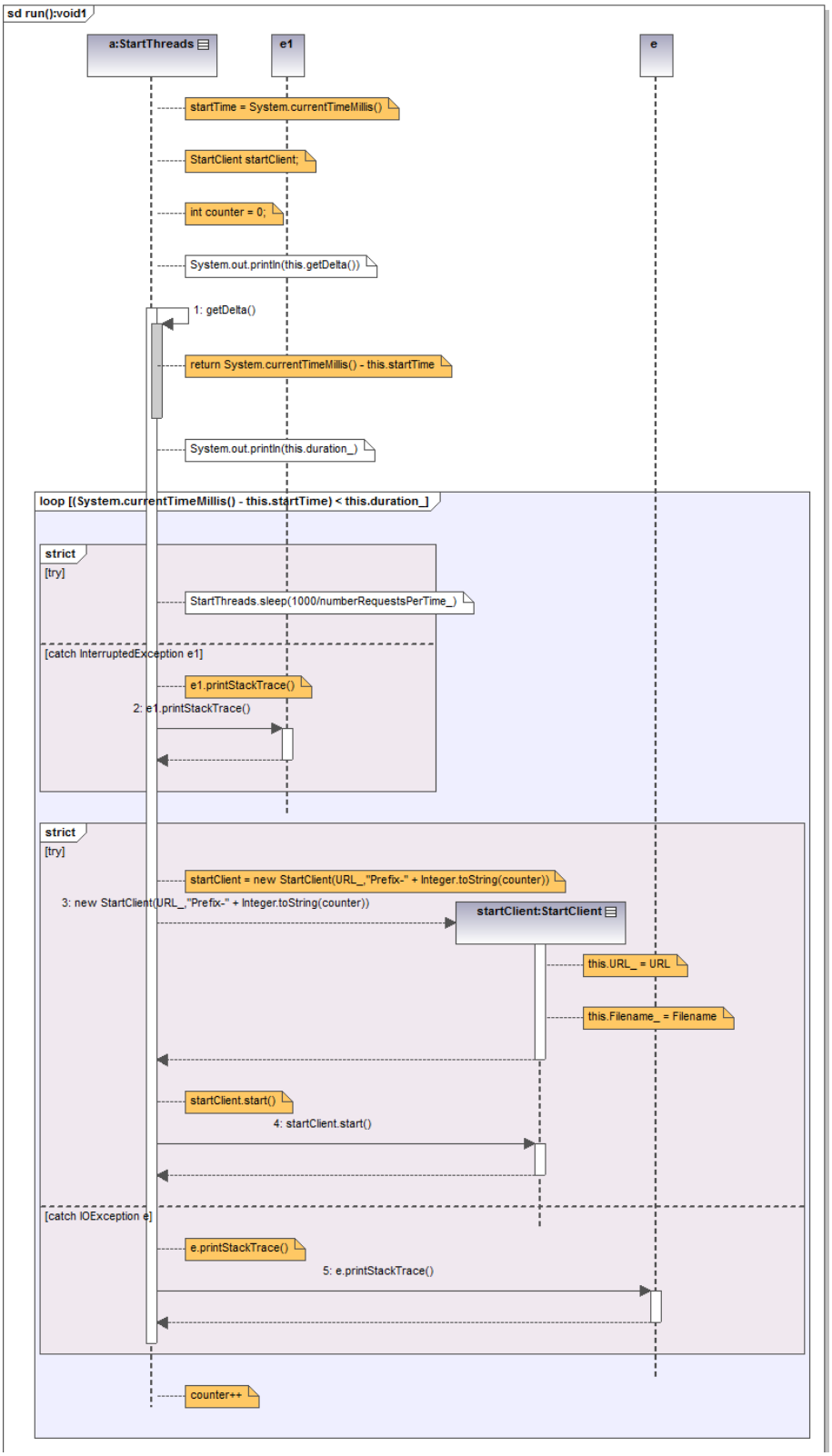

Figure B.2: Time Scheduling with Threads at Java Client Model

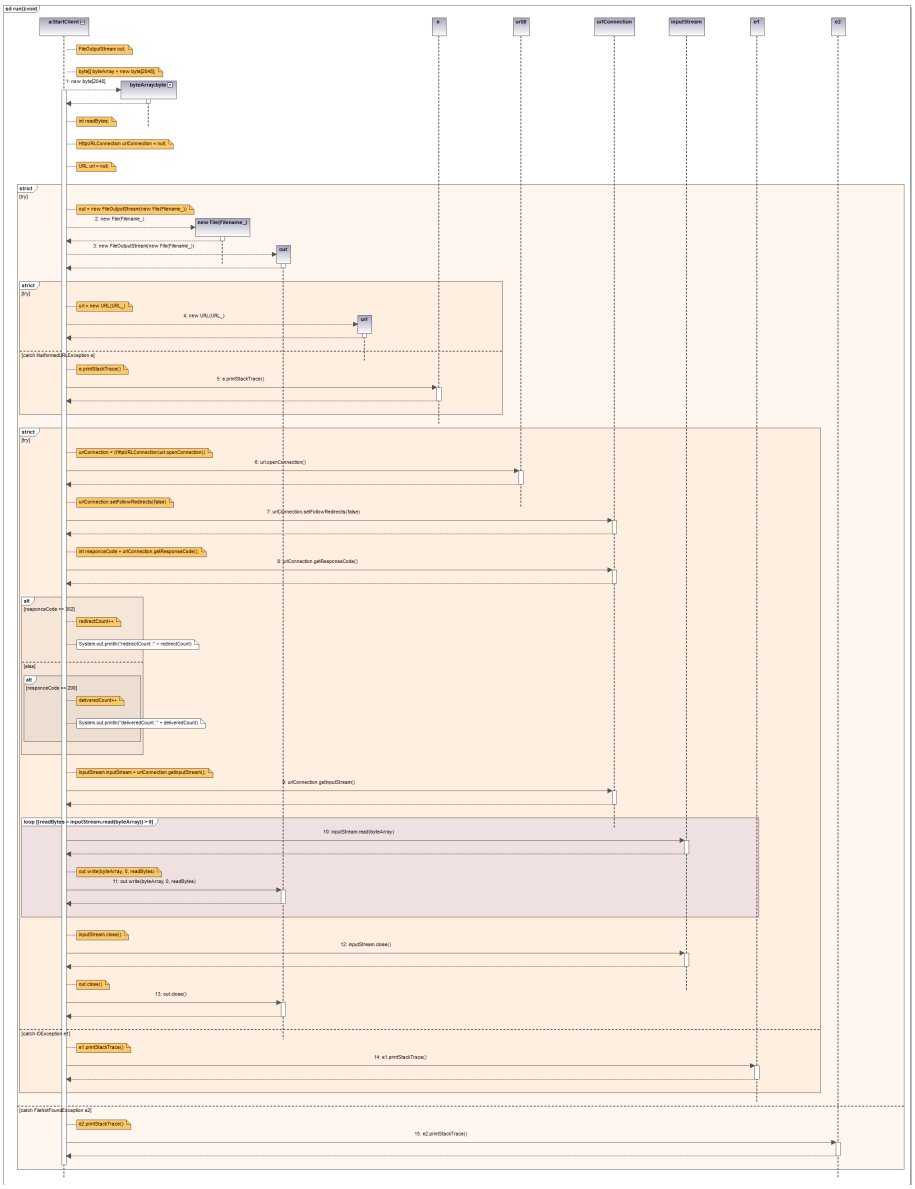

Figure B.3: Full Sequence Diagram of Java Client Model

Appendix C

Summary Log

Listing C.1: Example: Summary Log

```
Results of period #1 (from 2 sec to 7 sec ):
*****************************************
Completed Clicks: 46 with 0 Errors (=0.00%)
Average Click Time for 16 Users: 141 ms
Successful clicks per Second: 9.10
(equals 32'742.43 Clicks per Hour)

Results of period #2 (from 7 sec to 12 sec ):
*****************************************
Completed Clicks: 119 with 0 Errors (=0.00%)
Average Click Time for 32 Users: 6 ms
Successful clicks per Second: 23.44
(equals 84'364.81 Clicks per Hour)

Results of period #3 (from 12 sec to 17 sec ):
*****************************************
Completed Clicks: 201 with 0 Errors (=0.00%)
Average Click Time for 48 Users: 6 ms
Successful clicks per Second: 39.60
(equals 142'564.05 Clicks per Hour)

Results of period #4 (from 17 sec to 22 sec ):
*****************************************
Completed Clicks: 277 with 0 Errors (=0.00%)
Average Click Time for 64 Users: 6 ms
Successful clicks per Second: 54.53
(equals 196'318.51 Clicks per Hour)

Results of period #5 (from 22 sec to 27 sec ):
*****************************************
Completed Clicks: 353 with 0 Errors (=0.00%)
Average Click Time for 80 Users: 6 ms
Successful clicks per Second: 69.40
(equals 249'841.85 Clicks per Hour)

Results of period #6 (from 27 sec to 32 sec ):
*****************************************
Completed Clicks: 426 with 0 Errors (=0.00%)
Average Click Time for 96 Users: 8 ms
```

```
Successful clicks per Second: 83.69
(equals 301'291.22 Clicks per Hour)

Results of period #7 (from 32 sec to 37 sec ):
*****************************************
Completed Clicks: 504 with 0 Errors (=0.00%)
Average Click Time for 111 Users: 7 ms
Successful clicks per Second: 99.01
(equals 356'418.59 Clicks per Hour)

Results of period #8 (from 37 sec to 43 sec ):
*****************************************
Completed Clicks: 584 with 0 Errors (=0.00%)
Average Click Time for 127 Users: 7 ms
Successful clicks per Second: 114.00
(equals 410'409.38 Clicks per Hour)

Results of period #9 (from 43 sec to 48 sec ):
*****************************************
Completed Clicks: 662 with 0 Errors (=0.00%)
Average Click Time for 143 Users: 8 ms
Successful clicks per Second: 129.89
(equals 467'592.19 Clicks per Hour)

Results of period #10 (from 48 sec to 53 sec ):
*****************************************
Completed Clicks: 723 with 0 Errors (=0.00%)
Average Click Time for 159 Users: 7 ms
Successful clicks per Second: 141.07
(equals 507'866.46 Clicks per Hour)
```