Optimizing an Endicia Web Service

A Major Qualifying Project

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

by

Lauren Kahn, Dylan Kirby, David Mihal

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science

May 2015

Approved:

David Finkel, Advisor

ACKNOWLEDGMENTS

We would like to thank Patrick Farry and the rest of the team at Endicia for their help and support throughout this project. We would also like to thank our MQP Advisor, Professor David Finkel.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

ABBREVIATIONS

DNS	Domain Name System
DPI	Dots-per-inch
ELS	Endicia Label Service
HTTP	Hypertext Transfer Protocol
JPEG	Joint Photographic Experts Group *(image format)*
PDF	Portable Document Format
PNG	Portable Network Graphics *(image format)*
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol

ABSTRACT

Lauren Kahn, Dylan Kirby, David Mihal B.S., Worcester Polytechnic Institute, May 2015. Optimizing an Endicia Web Service. David Finkel.

Our project was focused on Endicia's Label Server, which provides web services for generating shipping labels. Our goal was to improve the performance of the Label Server at geographically diverse locations. To achieve this goal, we developed a proxy server that could be deployed to various cloud locations. This proxy received compressed vector graphics from Endicia's data center in San Jose, and sent bitmap images back to the client. Using our proxy server, we reduced the response time by approximately 50%.

1. INTRODUCTION

While computers allow for quick and easy communication, mail and shipping are still very important in today's society. The mailing industry is more than a one trillion dollar industry and the United States Postal Service only last year processed over 150 million mail pieces. [1] To meet this demand, new technologies try to make mailing and shipping an easier and more efficient process. Endicia is one prominant company that aims to produce these technologies. Founded in 1982, Endicia began as a consulting company but entered the postal industry with one of its early customers, the United States Postal Service. Today, Endicia is the leading provider of eCommerce shipping technologies. [2]

Our project was focused on Endicia's Label Server, which is used to generate shipping labels. Our goal was to improve the performance of the Label Server at geographically diverse locations. In order to achieve our goal, we were given the idea to develop a proxy server that would be deployed in various cloud locations to minimize the ammount of time taken by sending large files from the Label Server in San Jose, California.

2. BACKGROUND

2.1 Endicia

Originally founded as a technology consulting company in 1982 under the name PSI Associates, Endicia is a software company whose products are aimed towards simplifying the postal services. [2] Although their products have been in development since the founding of PSI, Endicia officially entered the software postage industry in 1989 (while still known as PSI Associates) with the release of the product Envelope Manager and the invention of Dial-A-ZIP Address Verification. PSI continued to release products for the postage industry until the year 2000, when a subsidiary of PSI Associates was formed to manage a product called the Endicia Internet Postage. Endicia has continued to produce different technologies for the shipping industry ever since.

Today the company is based in Palo Alto, California and creates software solutions to help take the hassle out of shipping for businesses and individuals. Endicia's main product is DAZzle, a software program that gives users access to all electronic services provided by Endicia. Additionally, Endicia releases a program called DYMO Stamps that provides limited functionality for users who do not want to pay for the complete coverage.

Endicia is partnered with a large number of companies in order to provide their software and hardware products. Three of the most notable are the United States Postal Service, Microsoft, and DYMO. In 2007 Endicia was acquired by Newell Rubbermaid, a company with branches in office supplies, household products, and hardware products. This merger allowed Endicia to partner with other Newell Rubbermaid acquisitions to provide hardware pairings with their software. DYMO, a subsidiary

of Newell Rubbermaid, fit this need with their brands of printers. [3] In 2015, more than twelve billion dollars in postage had been printed. [4]

There are a small number of other companies offering similar services to Endicia, the most prominent of them being Stamps.com. Stamps.com offers a variety of subscription-based plans, each one targeting different types of users, from the home office user to the enterprise user. Users can also purchase postage supplies directly from their website. [5]

2.2 Label Server

The Endicia Label Server (ELS) is a web service that provides customers with shipping labels on demand. The ELS Web service is called by various customer software applications such as online shopping carts and warehouse management systems. This service allows customers to easily integrate services from the US Postal Service into their existing systems. The server can generate shipping labels in various image or printer language formats such as JPEG, PNG, PDF and ZPLII (the printer language for Zebra thermal printers). In addition to generating shipping labels, the Label Server also includes other postal features such as address verification and postage rate calculation. [6]

2.2.1 Label Server Operation

Endicia's Label Server produces shipping labels based on users' requests. The requests contain the type of label, address information, delivery information, special services, and more. Once this request is received by the Label Server, address verification, validation, price checks and other necessary functions are performed. The requests are made using the SOAP protocol. SOAP is an XML-based messaging protocol that defines a set of rules for structuring requests. [7] The requests are formatted using XML and the server responds with XML containing the generated label image and other important information.

After the shipping label is received by the Label Server, the user must be authorized by the authorization engine. If authorization is not confirmed, the Label Server will not continue making the shipping label. After user confirmation, requests are sent out to get the price using the address, package type, and other necessary information that were sent in the request. After all necessary checks and calculations are made, the rendering engine will generate the shipping label. [5]

2.2.2 Rendering Engine

The rendering engine of Endicia's Label Server is responsible for creating the shipping labels after the label data has been collected. The rendering engine first loads the shipping label's layout template. There are several hard coded templates that can be chosen from based on the user's request. Each of these templates is an XML document that contains all of the components needed to create a label. These components include label fields, lines, rectangles, text boxes, the barcode, the indicium, and more.

Once the label layout has been chosen and loaded, the rendering engine binds the label data that was collected in the user's request to the label. Once all of the data binding has been completed, the shipping label is drawn onto a virtual canvas. The virtual canvas is then converted into the image format that the user requested, such as PNG, JPEG, or PDF. [5]

2.3 HTTP Network Performance

In order to improve the performance of Endicia's web services, we must first understand the different properties that affect network performance. The majority of web services, including Endicia's ELS, use HTTP as a transport protocol. We can analyze each segment of an HTTP request to understand factors that affect performance.

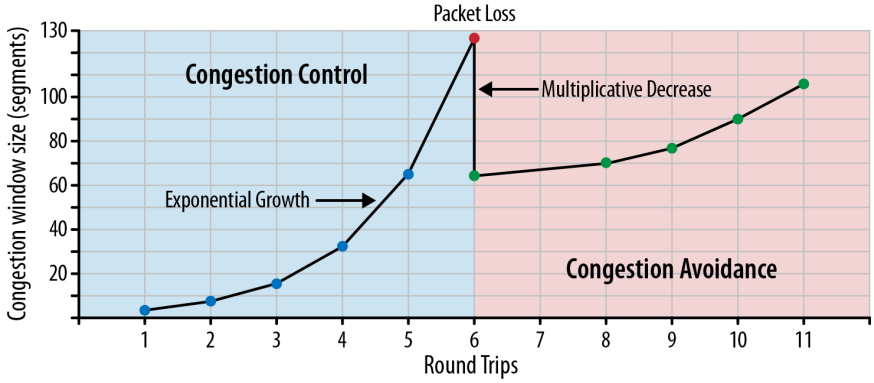

Figure 2.1. TCP slow-start transmission speed [9]

2.3.1 DNS Lookup

The Domain Name Service, or DNS, is a decentralized system for resolving hostnames to IP addesses. In other words, the DNS system allows computers to convert a human-readable name like "google.com" into a machine-readable IP address like "216.58.217.46". Most network requests begin with a DNS query. [8]

2.3.2 TCP Connection

TCP is the transport-layer protocol used by HTTP, as well as the majority of other application-layer protocols. TCP compensates for the unreliable nature of packet-switched communications and provides reliable data transmission. This is achieved through error detection, flow control and retransmission.

TCP provides congestion-control to avoid overloading a network with data. Congestion control is implemented using a algorithm called "slow-start". Using this algorithm, a TCP will client will initially send packets at a slow rate. As packets are successfully transmitted and acknowledged, the rate of package transmission slowly increases. If a significant number of packets are lost, TCP will substantially decrease the rate at which packets are sent. [9] This pattern can be seen in Figure 2.1.

This TCP slow-start process occurs whenever a new TCP connection is created. Therefore, older TCP connections will tend to be faster than recently created connections.

2.3.3 SSL Connection

Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) are cryptographic protocols used to provide end-to-end encrypted communication.

Use of SSL introduces some overhead into the communication. First, an SSL connection is established by exchanging cryptographic information, a process known as a "handshake". This handshake involves two round-trip communications. This means that using SSL can add a significant amount of time if the connection between the client and server has high latency.

2.3.4 Awaiting Response & Server Processing

After the TCP and SSL connections have been established, the HTTP client will transmit the HTTP request, which specifies the desired resource to access. If the client is sending a large amount of data in the request, it will take time to send the full body of the request. However, most HTTP requests tend to be small, so the upload time is negligible.

After the HTTP server receives the request, it must process it and generate a response. If the server is a simple web server, it will often only need to read a document from the disc and send that as the response. However, web services such as Endicia's ELS will often do more complicated computations on the server and access other resources such as databases.

2.3.5 Response Download

Once the server generates a response, it will send it back to the client. Responses will often be substantially larger than the request, as they will contain requested documents or images. If the response is large, the time for the response to download will be limited by the bandwidth of the network between the client.

2.4 Cloud Services

Cloud computing, often referred to as simply "the cloud", is the delivery of on-demand computing resources–everything from applications to data centers–over the Internet on a pay-for-use basis. Cloud-based applications run on distant computers in the cloud that are owned and operated by others and that connect to users' computers via the Internet. [10] There are many benefits to using cloud computing as opposed to maintaining your own physical hardware. Using these services typically reduces capital costs, as there's no need to spend money on hardware, operating systems, and maintenance personnel. These services can be deployed around the world at a number of different regions and requires minimal setup. This makes it easy to scale up computing resources.

There are many different companies providing various cloud services. Currently, two of the most popular services are Microsoft Azure and Amazon Web Services (AWS).

2.4.1 Microsoft Azure

Microsoft Azure is a cloud computing platform and infrastructure for building, deploying and managing applications and services through a global network of Microsoft-managed datacenters. [11] Azure provides services for hosting websites, deploying virtual machines, setting up web services, and cloud file storage. Azure's file storage service is charged by the amount of data stored, while Azure's other services charge

on a monthly or hourly basis depending on the amount of resources allocated. For example, cloud services are billed by the number of CPU cores and the size of memory allocated for the service. Microsoft Azure provides a free 30 day trial, with a $200 credit towards their services.

2.4.2 Amazon Web Services

Amazon Web Services (AWS) is a cloud computing platform owned by Amazon.com that provides a collection of remote computing services. [12] The most central and well-known of these services are Amazon Elastic Compute Cloud (EC2) and Amazon (Simple Storage Service) S3, which provide cloud computing and data storage, respectively. As with Microsoft Azure, you pay a fee relative to the amount of data stored or per hour of EC2 virtual machine usage. These fees increase according to the amount of data stored and the resources allocated to virtual machines. AWS provides a free tier for their services, which includes up to 750 hours worth of EC2 computations, and 30GB of free file storage. These limits are applied per month, and new customers can remain on the free tier for up to a year.

3. REQUIREMENTS

Endicia provides services to many different companies across the United States, as well as around the world. Currently, all customers access Endica's services by communicating with servers at a data center in San Jose, CA. This is perfectly adequate for most customers in the western United States, however service performance decreases as clients get farther away from San Jose.

Endicia hoped to improve the performance of these services by providing more globally-distributed infrastructure. The simplest way to provide distributed infrastructure would be to deploy the ELS to across the globe in cloud-hosted virtual machines. Unfortunately, the requirements of the ELS make it impossible for this type of deployment. Endicia's close integration with the U.S. Postal service requires them to use extra security with their data. This is achieved by using hardware cryptography devices in the servers. Cloud computing providers do not allow access to the physical hardware running these servers, so it is impossible to deploy the whole ELS server to the cloud.

In order to improve the performance of Endicia's existing infrastructure, we were asked to explore a hybrid-cloud model. The hybrid-cloud model is where a system is made up of multiple types of cloud systems, or in our case, integrating cloud services with traditional centrally-managed infrastructure.

Our cloud-deployed servers would improve performance by allowing customers to connect to a proxy server close to them, instead of having to connect to the servers in San Jose. The proxy servers then pass the requests from the client to the main ELS server, and forward responses back to the client. The performance increases come from the proxy handling some computation, allowing the ELS server to send smaller responses.

One of the most main service calls made to the ELS server is the `GetPostageLabel` request. This method allows a user to make a request for a shipping label, and they will receive a response that includes an image of a postage label that can be printed. These images are typically sent in bitmap formats such as PNG or JPEG, which have sizes around 160kB and 300kB, respectively. The project description from Endicia's internal wiki describes the performance issues with sending these large bitmaps over large distances.

> "The response time of our API depends on both the label format and the location of the customer relative to Endicia. The bitmap based formats are larger than equivalent vector graphic formats by a factor of 10:1. We have anecdotal evidence that ZPL labels (a type of vector graphic format) can be received in Atlanta in 400ms, whereas the same label in PNG format may take 1.5 seconds or longer. To see the impact of location, the same label in PNG format can be retrieved in 360ms from Northern California. We have customer in Europe and in Asia for whom a label may take several seconds to be received." [13]

Based on this data, it should be substantially faster to send light-weight data between the ELS serve and our proxy, and only sent the large bitmap the short distance from the proxy to the client.

To avoid having to reconfigure clients, the proxy should use the same API as the existing ELS server. For `GetPostageLabel` requests, the proxy would change the image type of the request to reduce the amount of data to be communicated with the ELS server. Once the proxy receives this response, it should use information from the response and request to generate the label image and send it back to the client.

4. METHODOLOGY

Before implementing the proxy, we first set out to evaluate the existing performance of the Label Service. This provided a baseline performance to try to improve on, and would highlight regions in which a proxy server would be beneficial. Next we developed a some simple proxy servers in order to evaluate the intended performance gains. We then built a test environment to simulate requests to this proxy from different regions of the world. Finally, we developed the proxy servers and deployed them to different regions around the world using cloud services, and evaluated their performance.

4.1 Evaluating current system

First we evaluated the current performance customers are experiencing with requesting labels. We reviewed data from two different sources: Keynote and Endicia's server logs. This data provided a baseline for the proxy server to improve on, and can highlight areas that would benefit from a proxy server.

4.1.1 Keynote Timing Data

Keynote is a service that partners with ISPs around the world to report website performance. Endicia has set up Keynote to make periodic API requests to its services to see how they perform in different areas. We collected data from Keynote reports to analyze round-trip response times for requesting labels in different regions. An example of this geographic performance data can be seen in Figure 4.1.

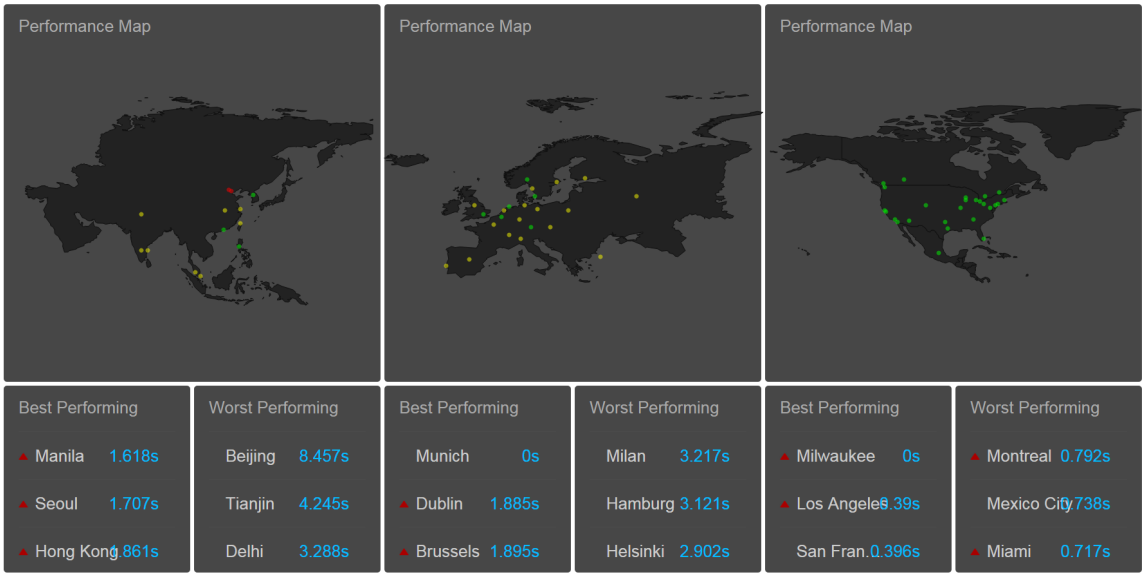

Figure 4.1. Service response times by geographic region, provided by Keynote

This service also provided data on how much time was spent on each segment of the network connection. This gave us the parts of the response we should work on reducing.

4.1.2 ELS Server Logs

The Endicia Label Server records a log entry for each request made. We were given a set of log entries recorded over one day. These log entries record data such as the API called, requester's location, response time, timestamp, label format, and other parameters. We first filtered the entries to only those requesting labels from the `GetPostageLabel`. We then queried this set of log entries to evaluate the response times for clients in different regions and response times for different label formats. From these server logs, we were able to see that the response times were longer than desired and in what locations in particular the response times was poor.

4.2 Develop Evaluation Model

The next step was to develop a test environment to simulate requests coming from different regions using a proxy, without fully developing the proxy. The results of developing the proxy server simulation are discussed in Section 5.1.

4.2.1 Gather Network Parameters

In order to simulate long-distance requests to the ELS service, we need to understand the network performance of distant customers. There are two main parameters we looked at for network performance: latency (delay to send data in relation to distance) and bandwidth (data transfer rate). To gather an estimate for bandwidth we used a website SpeedTest [14]. This tool performs an upload and download to a server in a region of your choice and returns an estimated bandwidth of your connec-

tion. We ran this tool from WPI and Endicia to estimate the bandwidths of their connections.

To gather latency estimates for different regions we used an online reference called WonderNetwork [15]. This website performs pings from different regions around the world to other regions and maintains an average ping time reported. We used this to gather latencies for San Jose to Boston, San Jose to London, and San Jose to Hong Kong. This allowed us to simulate label requests from coast to coast in the US, from Northern Europe, and from Southern Asia.

4.2.2 WAN Simulation

To simulate a network with these parameters, we needed a WAN emulator. After evaluating multiple tools we decided to use SoftPerfect Connection Emulator [16], which is easy to install on Windows and provides a nice interface to simulate different bandwidths, latencies, and other parameters. More details about SoftPerfect Connection Eumulator are provided in Appendix A.1.

4.2.3 Performance Evaluation

We used SoftPerfect Connection Emulator in parallel with SoapUI to make API calls to ELS under desired network conditions. SoapUI is an interface used to make SOAP API calls, and returns a response and round-trip response time. More details about SoapUI are provided in Appendix A.2. The resulting response times could be compared to direct calls to the API to validate that our model accurately simulated requests coming from the east coast. For our trials, we requested ten labels for each set of desired network conditions and recorded the response time as the median of the last eight requests. We skipped the first two results because sometimes it could take time to warm up, and we took the median of the remaining calls because occasionally there were outliers that took longer than average. To get a baseline to compare to,

we requested the same labels from the clients to the ELS server. These results could also be compared to the data collected from customer and Keynote requests.

4.3 Simple Proxy Development

We developed the proxy server in increments. We first developed a proxy that simply forwarded the request to ELS, then had the proxy demonstrate it could modify the request and response, then had it return a hardcoded image, and then finally had it fully draw the label requested. We wrote the proxy servers in C# and started by running them as a service on our machines at Endicia. We also set up a Windows virtual machine using Virtualbox [17] to act as the client.

4.3.1 Request Passthrough

The first proxy server we developed simply acted as a middleman to the ELS server. This simple proxy server would take in the same `GetPostageLabel` requests and forward them to ELS, get a response back, and return the response to the client. This obviously added some time to the response time but it was a good starting point for making a more efficient proxy.

4.3.2 Payload Modification

The next step was to show that our proxy could change the request made, instead of simply passing it through. The ultimate goal was to change the request to fetch a smaller image, and then draw the requested format at the proxy. To illustrate we could modify the request like this, we modified the proxy to request a label of type EPL2 (around 3.5kB) no matter what label format was requested. The proxy would then return the label in EPL2 format, with the image modified to include a red dot to illustrate the image was manipulated by the proxy.

4.3.3 Hard-coded Images

The next step was to have the proxy return the label in the format requested. Rather than go straight to drawing the label, we instead had the proxy return a hard-coded image of the label format requested. A client could request a label of type PNG or JPEG, the proxy would then request an EPL2 label from ELS, and then return a hard-coded PNG or JPEG back to the client. The label being returned would be roughly the same size as it should be, but lacked the component to draw the real fields of the label requested. Using this proxy we were able to get a good estimate of how the final product would perform.

4.3.4 Fully Functional Proxy

The last step of developing the proxy server was to separate the drawing component from ELS and perform the drawing on the proxy instead. Rather than returning a hard-coded image, this proxy would get the EPL2 label back and convert it to the format requested. At this stage the proxy was fully functional and we could simulate how it would perform in real-world situations. The proxy acted just as the ELS API would, and returned errors if users make invalid or unauthorized requests.

4.4 Develop Label Drawing Component

After getting the proxy server working, out next step was to take the drawing functionality out of the existing label server code in order to draw the label on the proxy server. Our first step was to change certain fields on the label and see the change in the output. After we had that basic version of the drawing component working, we added the functionality to draw all the required fields and to draw different types of labels. Our initial approach was to modify the existing drawing code. Our second approach was to use the ELS label format and parse a text file to draw the label.

4.4.1 Modifying the Existing Drawing Code

Our first approach to move the drawing functionality to the proxy server was to modify the existing drawing code. Our first step was to remove all calls to databases, servers, engines, etc. since the whole purpose of moving the drawing component to the proxy server was to have only one call to the proxy so that data did not have to travel as far. While we got this approach to draw the whole label except the barcode and the indicium, it was very messy code since it was a lot of copied code that was modified for our purposes in many different files. This made the code hard to follow and understand although we were able to get the label image almost complete which was our goal. However, when we were trying to add the barcode and the indicium drawing parts, a different approach was found that seemed a lot simpler that involved parsing a text file for drawing instructions.

4.4.2 Drawing the Label using the ELS Label Format

Our second approach was to use the ELS label format to get the instructions to draw the label rather than having to create the instructions through the old drawing code. The ELS label format returns a list of drawing instructions, such as `DrawLine`, `DrawString`, and `DrawBarcode`, in a text file. More details about the ELS label format are discussed in Appendix B. Therefore, in order to draw the shipping label from these instructions, we had to parse the text file and then call the drawing functions with the parsed values in the order they were listed in the ELS file.

4.5 Deploy Proxy Servers

Now that we had the proxy servers running under simulated network conditions, the next step was to deploy them to actual locations around the world as they would be used in real scenarios. We chose to deploy the proxy servers to Microsoft Azure data centers, and nearby clients to Amazon Web Services. The proxies could be

deployed anywhere, but we decided to demonstrate this with Microsoft Azure as it was easy to export a Visual Studio project to a Microsoft Azure cloud service. The results of the proxy server deployment are discussed in Section 5.2.

4.5.1 Cloud Environments Setup

US Servers

Around 27% of label requests coming from the US originate from the east coast. To evaluate requests coming from the east coast of the US, we set up a proxy instance with Microsoft Azure in Virginia. For a client we continued to use a WPI machine to make requests to this proxy server.

Europe Servers

Around 30% of all foreign label requests originate from Europe. To evaluate requests going to a proxy server in Europe, we set up a proxy instance with Microsoft Azure in Ireland. For a client we set up a client with AWS, also in Ireland.

Asia Servers

Around 23% of all foreign label requests originate from Asia. To evaluate requests coming from Asia, we set up a proxy instance with Microsoft Azure in Hong Kong. For a client we set up a client with AWS in Singapore.

4.5.2 Performance Evaluation

With the proxy servers deployed globally, we could now evaluate the performance of them in real world scenarios. On each of the client machines we ran two trials: one set of label requests to the ELS API in PNG and JPEG formats, and one set of label requests to the nearest proxy server in PNG and JPEG formats. By comparing these

two results we could see if the proxy servers decreased the roundtrip response time significantly. We continued to have the clients use SoapUI to request the labels.

5. RESULTS

Our results include the performance of both the simulated proxy server, as well as the final deployed proxy server. Our results for the simulated proxy follows the methodology discussed in Section 4.2, while the final deployed server follows the methodology discussed in Section 4.5.

5.1 Simulated Proxy Server Performance

Using our test environment we simulated requesting labels from distant locations using just our machines at Endicia. We used the network emulator to simulate requests coming from the eastern United States, northern Europe, and southeast Asia. We requested PNG and JPEG shipping labels from the Label Server directly and also through our proxy server returning hardcoded labels. Figure 5.1 shows the performance of our proxy server compared to requesting labels directly. The graph shows the average response times to generate a label of each type, coming from each simulated region. The results of this simulation indicate that the proxy server would improve the performance of requesting labels. For each simulated region we saw a decrease in response time of 50-60% using the proxy servers.

5.2 Proxy Server Performance

After the proxy server was fully developed we deployed instances of it to the eastern United States, northern Europe, and southeast Asia. We ran the same performance evaluation with these new proxy servers, with requests coming from nearby clients hosted in the cloud. The results of the performance evaluation are shown in Figure 5.2. The graph shows the average response times to generate a label of

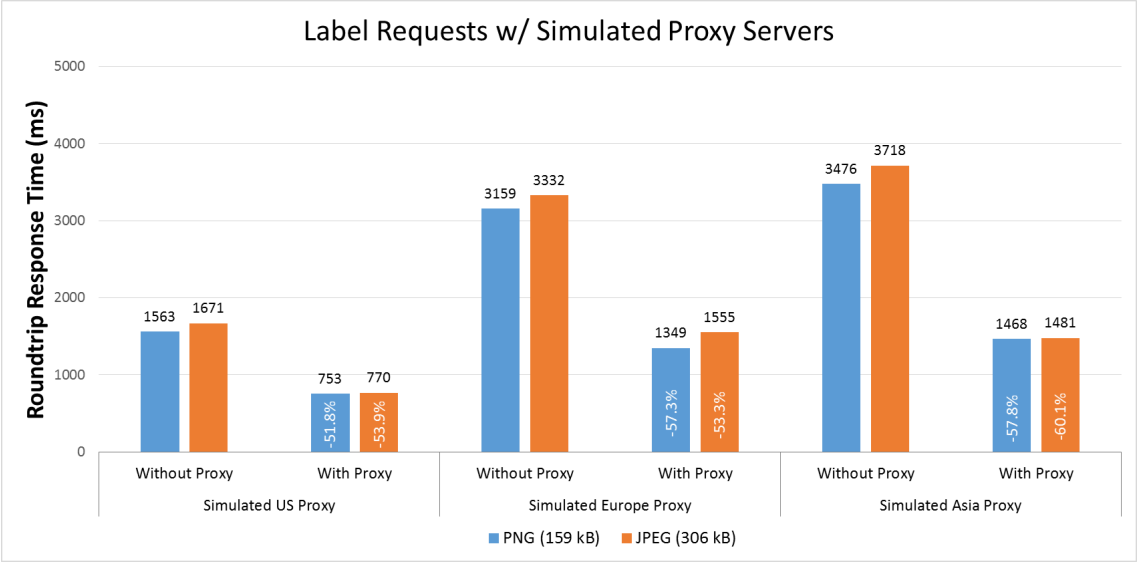

Figure 5.1. Simulated Proxy Server Performance

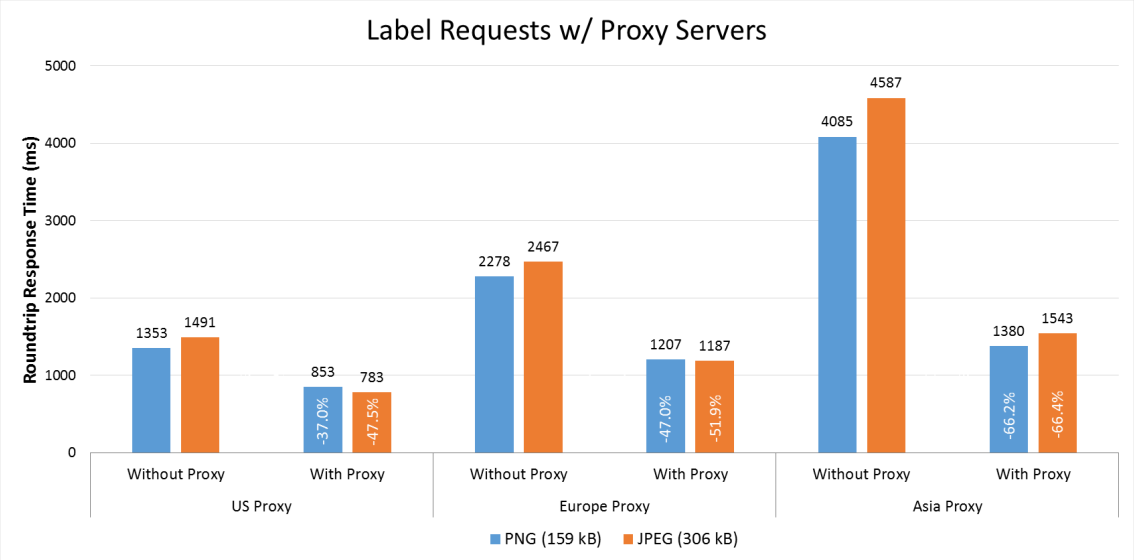

Figure 5.2. Proxy Server Performance

each type, coming from each region. The results are similar to the simulated proxy results-in each region we saw a response time reduction of 35-65%. The response time reductions are most significant for long-distance clients. In southeast Asia, labels that used to take almost 5 seconds to generate were generated in under 2 seconds using the proxy server. Overall these results show that distant clients will be able to produce about twice as many labels in a day by connecting to a nearby proxy server.

We also investigated the performance of a proxy for a customer that is hosted in the same data center as the proxy. We tested this by setting up a client and a proxy server both in the same Microsoft Azure region. We then requested labels from the client directly to the Label Server and compared the response times to requesting labels from the client going through the proxy. The results of our evaluation can be seen in Figure 5.3. You can see that it would be beneficial for customers hosted in a data center to connect a proxy that is also in the same data center. The response times decreased by about 40% by connecting to the proxy server instead of directly to the Label Server.

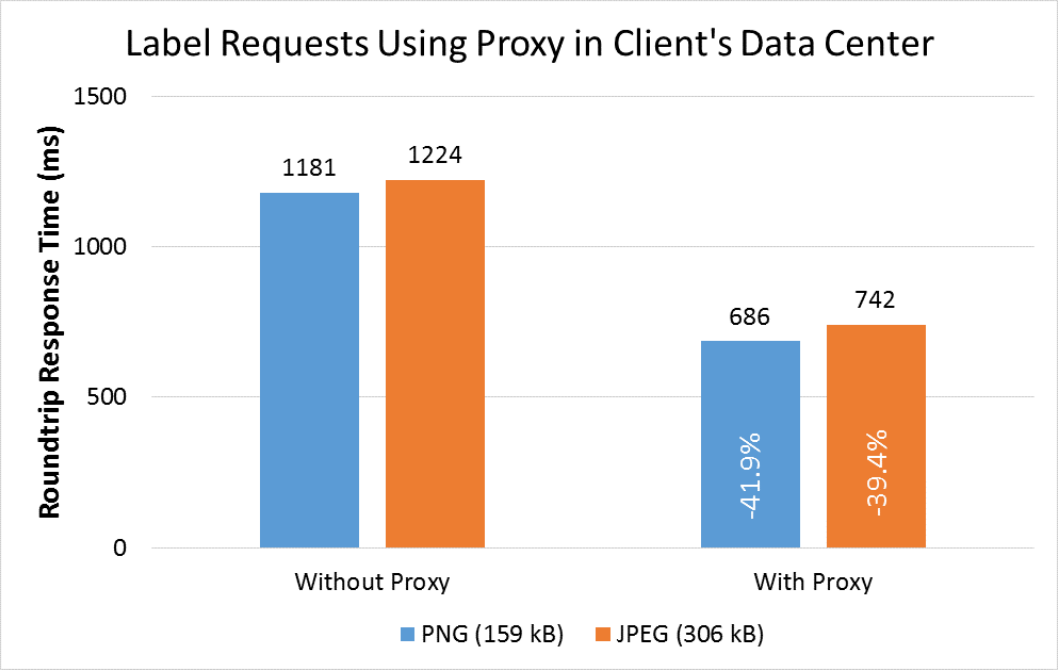

Figure 5.3. Label Requests Using Proxy in Client's Data Center

6. CONCLUSIONS

During our project, we were able to successfully build a proxy server to draw shipping label images. Our tests show that by using this proxy, response times are typically reduced by approximately half.

Towards the end of our project, we were able to meet with the representative of one of Endicia's partners. Their company primarily provides shipping labels to companies in China, so they often experience the slow performance that results from sending images across the globe. The representative was very interested in our proposed solution, which allowed us to see first hand how much value our project could offer to Endicia and their global customers.

Another outcome of this project is the research into setting up a simulated test environment. It was very useful for our project to be able to run tests on our local machines that still accounted for network limitations such as high latency and low bandwidth. Using the tools and techniques employed in our tests, Endicia employees could easily set up their own test environments to simulate the network performance of other services.

Our work in setting up cloud services for Endicia also has many other practical applications. Being able to quickly set up these ELS proxy servers at data-centers across the world can allow Endicia to quickly scale up to meet demand and provide network solutions targeted towards specific customers. With the growing popularity of cloud computing, this introduction to cloud services is a substantial step for Endicia's constantly evolving technology stack.

6.1 Future Work

While speeding up the time to generate shipping labels by around fifty percent is a big achievement, there are additional improvements that can be made to the system to add both more functionality and more speed improvements.

One way to add additional functionality is to store logo images on the proxy server so they do not have to be fetched from the Label Server each time a shipping label with a logo needs to be printed. This would be a good improvement because most companies use the same logo every time they need to print a shipping label. Therefore having to get the same logo image every time from the server is a waste of time.

Another feature that would help make the proxy server even more helpful would be to store the postage rate table and any other static information that does not change frequently on the proxy server. By storing this information on the proxy server, less data would need to be fetched from the label server each time a new request for a shipping label is made which would increase the performance.

APPENDICES

A. TOOLS

In order to see how the proxy would affect the the performance of the Endicia Label Server, we needed tools to model network performance and label creation. In order to model different aspects of a real network, such as latency and bandwidth limits, we chose to use SoftPerfect Connection Emulator. To test the Label Server's web performance, SoapUI was used and to test how the label looked after label creation, the ELS Test Client was used.

A.1 SoftPerfect Connection Emulator

SoftPerfect Connection Emulator is a Wide Area Network (WAN) Emulator for Windows. This software helps software developers create network-enabled applications, especially those that are time-critical, that need to be tested in a range of different environments. SoftPerfect Connection Emulator imitates a network connection with various bandwidth limits, delays, losses, and other problems. This allows software developers to ensure the quality of their application regardless of how good or bad the connection.

We chose to use SoftPerfect Connection Emulator because of several factors. One factor was that it runs on any PC with Windows XP or higher and that it is very easy to install. Another factor is that after installing the software, it is very easy to use and get started using quickly. An additional factor is that it had all of the necessary features we needed to simulate the network, including latency and bandwidth. This tool can also be ran from the command line with a given file for the network settings in XML format. This makes it easy to automate different network configurations while making requests to the Label Server.

A.2 SoapUI

SoapUI is a cross-platform solution for testing web services through a graphical interface. The application allows you to easily create and execute automated functional, regression, compliance, and load tests. For our purposes we use SoapUI to make SOAP requests to the Label Server. By performing the requests through this interface we can easily see the returned response for further analysis as well as the response time taken to complete the request. By using this tool in parallel with SoftPerfect Connection Emulator, we can quickly make subsequent requests to the Label Server under different network conditions, log the responses and response times.

B. ELS LANGUAGE SPECIFICATION

This secton contains a formal definition of the ELS file format returned by the Endicia Label Service. ELS is a non-standard file format used to to describe the label images returned by this service.

The ELS format does not contain data to represent a raster image. The file is a text document containing metadata and instructions needed to draw the image. The set of instructions roughly correspond to the methods in the `BaseCanvas` class that are called to draw the image.

B.1 ELS Header

`ELS`
`Dpi,` *(dpi)*
`Width,` *(width)*
`Height,` *(height)*
`Rotation,` *(None, Rotate90, Rotate180, or Rotate270)*
`BeginForm`

The header of an ELS document starts with a line reading `ELS`. The dimensions of the image are then provided, such as the height and width of the document (in inches), the resolution of the document in dots-per-inch, and the rotation of the documetn.

B.2 ELS Body

`BeginForm`
Drawing Commands
`EndForm`

The body of an ELS document consists of a series of commands between the `BeginForm` and `EndForm` lines. Each command takes 1 line, with the exception of the `Text` command which specifies the number of additional lines that are used.

B.3 Drawing Commands

B.3.1 `Clear`

`Clear`, *(color)*

The `Clear` command clears the whole canvas and sets it to the provided color

B.3.2 `ResetTransform`

`ResetTransform`

B.3.3 `TranslateTransform`

`TranslateTransform`, *(x)*, *(y)*

B.3.4 `FilledRectangle`

`FilledRectangle`, *(x)*, *(y)*, *(width)*, *(height)*, *(color)*

The `FilledRectangle` command draws a rectangle, with the top left corner at the provided coordinate and with the specified height and width.

B.3.5 `Rectangle`

`Rectangle`, *(x)*, *(y)*, *(width)*, *(height)*, *(pen width)*, *(color)*

The `Rectangle` command works similar to the `FilledRectangle` command, with the difference being it draws a line rectangle as opposed to a filled shape.

B.3.6 `Text`

`Text`, *(x)*, *(y)*, *(width)*, *(height)*, *(font name)*, *(font size)*, *(font style)*, *(color)*, *(alignment)*, *(line alignment)*, *(noClip)*, *(noWrap)*, *(vertical text)*, *(angle)*, *(number of lines)*
Lines of text to print

B.3.7 `Barcode`

`Barcode`, *(barcode type)*, *(x)*, *(y)*, *(width)*, *(height)*, *(center barcode)*, *(rotation)*, *(module width)*, *(barcodeHeight)*, *[barcode options]*, *(barcode data)*

The `Barcode` command draws a barcode at the specified location. The *barcode type* parameter specifies the type of barcode to draw, and accepts the following options:

- `GS1-128`
- `Code128`
- `Code39`
- `DataMatrix`
- `Postnet` *(Postal Numeric Encoding Technique)*
- `IMB` *(Intelligent Mail Barcode)*

B.3.8 `Line`

`Line`, *(x1)*, *(y1)*, *(x2)*, *(y2)*, *(pen width)*, *(color)*

The `Line` command draws a line between two points.

B.3.9 `Image`

`Image`, *(x)*, *(y)*, *(width)*, *(height)*, *(image format)*, *(image data)*

B.3.10 `FilledEllipse`

`FilledEllipse`, *(x)*, *(y)*, *(width)*, *(height)*, *(color)*

B.3.11 `Ellipse`

`Ellipse`, *(x)*, *(y)*, *(width)*, *(height)*, *(pen width)*, *(color)*

LIST OF REFERENCES

LIST OF REFERENCES

[1] U. S. P. Service, "United states postal service size and scope," 2015. https://about.usps.com/who-we-are/postal-facts/size-scope.htm.

[2] Endicia.com, "Company history," 2015. http://www.endicia.com/about-us/company-history.

[3] M. Tsai, G. Fleischer, and M. Alvarado, "Endicia webmailer application," 2011. http://www.wpi.edu/Pubs/E-project/Available/E-project-042711-140721/unrestricted/Endicia-Final-Paper.pdf.

[4] Endicia.com, "About us," 2015. http://www.endicia.com/about-us.

[5] E. Baicker-McKee, B. Pham, and J. Zhang, "Endicia label editor," 2012. http://www.wpi.edu/Pubs/E-project/Available/E-project-042312-172719/unrestricted/EndiciaLabelEditor_MQPReport.pdf.

[6] Endicia.com, "Endicia label server api," 2015. http://www.endicia.com/Developers/LabelServer.

[7] Soapuser.com, "Soap basics 1 : What is soap?," 2015. http://www.soapuser.com/basics1.html.

[8] D. Karrenberg, "Dns root name servers explained for non-experts," 2006.

[9] K. Fall and W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Professional Computing Series, Pearson Education, 2011.

[10] IBM, "Ibm cloud computing: What is cloud computing?," 2015. http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html.

[11] M. Azure, "Microsoft azure: Cloud computing platform & services," 2015. http://azure.microsoft.com/en-us/.

[12] I. Amazon Web Services, "Cloud computing services," 2015. http://aws.amazon.com/.

[13] Endicia, "Endicia internal wiki," 2015.

[14] Speedtest.net, "Speedtest.net by ookla - the global broadband speed test," 2015. http://speedtest.net.

[15] Wondernetwork.com, "Wondernetwork," 2015. https://wondernetwork.com/pings.

[16] Softperfect.com, "Softperfect wan connection emulator for windows," 2015. https://www.softperfect.com/products/connectionemulator/.

[17] Virtualbox.org, "Oracle vm virtualbox," 2015. https://www.virtualbox.org/.