Bachelor Degree Project


UNIVERSITY OF SKÖVDE

# HTTP Load Balancing Performance Evaluation of HAProxy, NGINX, Traefik and Envoy with the Round-Robin Algorithm

Bachelor Degree Project in Science
with a major in Informatics
G2E, 30 ECTS
Spring term 2022

Alfred Johansson

Supervisor: Johan Zaxmy
Examiner: Thomas Fischer

# Abstract

Operating a popular website is a challenging task. Users not only expect services to always be available, but also good performance in the form of fast response times. To achieve high availability and avoid performance problems which can be linked to user satisfaction and financial losses, the ability to balance web server traffic between servers is an important aspect.

This study is aimed to evaluate performance aspects of popular open-source load balancing software working at the HTTP layer. The study includes the well-known load balancers HAProxy and NGINX but also Traefik and Envoy which have become popular more recently by offering native integration with container orchestrators. To find performance differences, an experiment was designed with two load scenarios using Apache JMeter to measure the throughput of requests and response times with a varying number of simulated users.

The experiment was able to consistently show performance differences between the software in both scenarios. It was found that HAProxy overall had the best performance in both scenarios and could handle test cases with 1000 users where the other load balancers began generating a large proportion of failed connections significantly better. NGINX was the slowest when considering all test cases from both scenarios. Averaging results from both load scenarios excluding tests at the highest, 1000 users, concurrency level, Traefik performed 24% better, Envoy 27% better and HAProxy 36% better compared to NGINX.

# Table of Contents

# 1 Introduction

The number of websites on the internet is constantly increasing and more is expected from the services they offer. To handle the large amount of traffic popular websites sees, they need a scalable and good performing solution to distribute traffic amongst several web servers.

One existing solution to this is to use a Hyper Text Transfer Protocol (HTTP) load balancer that is able to inspect the traffic on the application layer and act as a reverse proxy to distribute the load between several backend web servers. Two traditionally popular open-source software products with these features are HAProxy and NGINX (Nemeth et al., 2017) and two emerging alternatives are Traefik and Envoy (Shah et al., 2019). The latter two are both designed with container orchestrated services in mind and natively offers features to automatically generate routes for container-based microservices. Containerization itself is an emerging and powerful technology offering scalability, high efficiency and fast deployment (Watada et al., 2019) and it is used increasingly by tech giants to build businesses (Shah et al., 2019). The high degree of integration with container-based services makes Traefik and Envoy an attractive choice with potential to simplify administration in container orchestrated environments.

Setting the native integration with container orchestrators aside, performance is an important factor for the user experience and economic growth on the web (Arapakis et al., 2021). This study will perform an experiment to compare performance between HAProxy, NGINX, Traefik and Envoy when used as load balancing tools for HTTP traffic outside a container-based environment. The four load balancers will be given the same conditions in a virtual environment were latency, request throughput and error rates will be measured in load scenarios with a varying number of users to indicate performance.

The report is divided in 9 chapters. Following the introduction, chapter 2 will provide background information on load balancing systems and go through some previous studies on the topic. In chapter 3, motivations for the study as well as its aim, limitations and objectives will be stated. Chapter 4 concerns methodology including planning and designing the experiment and considering validity threats. Chapter 5 goes through the preparation and execution of the experiment as well as addressing the validity of the collected data. In chapter 6, the results from the experiment are presented. In chapter 7 conclusions are drawn from the results to answer the research question. Chapter 8 will contain a discussion including limitations, ethical and societal aspects and will be followed by chapter 9 where future work is discussed.

# 2 Background

This chapter provides background to help understand the research question and background on the evaluated load balancing software packages. An overview of previous scientific work in the area is also presented in section 2.7.

## 2.1 Load Balancing Concept

Popular websites need to be able to serve a large number of simultaneous clients. Users also expect the site to always be available and that the server responds with low latency. In IT the term *High availability* (HA) describes systems that are continuously operational providing services which are available to users to a large extent (de la Cruz & Goyzueta, 2017). Nemeth et al. (2017) claims that it is impossible to use a single server to run a highly available website for two main reasons: First, there is always a risk of unexpected downtime combined with the problem that there will be no way to take the server down for planned maintenance while keeping the service up. Second, a single server does not scale very well and will be more vulnerable to attacks and load spikes leaving the risk of downtime due to overload. If one server is not enough, the solution will be to use multiple servers. However, this introduces a new problem: A tool is needed to distribute incoming traffic amongst the servers. Such tools exist and are called load balancers. The general idea of a load balancer is that it processes incoming requests and distributes the work to other concurrently running independent systems (Membrey et al., 2012). This way, the heavy load of serving websites can be shared among as many web servers as required. The load balancer acts as a frontend, doing the lighter work of distributing the traffic. To achieve HA in real world scenarios, it may be necessary to have multiple load balancers or implement several layers of load balancing but the principle stays the same. Similar principles can be applied when it comes to performance which can be improved by scaling up the of available resources beyond the capabilities of a single server.

## 2.2 Load Balancing Methods and Algorithms

One very simple form of load balancing for web server traffic is to make use of Domain Name System (DNS). When browsing the web, DNS is used to translate domain names to IP addresses which is needed in order to make the actual connection to a web server. Many DNS servers are in fact capable of load balancing. One example is BIND that offers this type of load balancing by using multiple records for the same domain name. The DNS server then alters the response so that the first IP will be rotated between requests (Internet Systems Consortium, 2022). DNS load balancing may be the simplest type since it is easy to implement and makes use of existing infrastructure but it does not offer features that are found in other types of load balancers. Membrey et al. (2012) lists three main issues with DNS load balancing:

1. Stickiness: In many cases, especially with websites providing dynamic content, it is desirable that a client always is connected to the same server. HTTP which is the protocol used for web traffic is by itself stateless which is one reason why many websites use cookies. A cookie is a small file which can store identifiers and sessions between the client and server. Such, for example, session cookie will only be valid between the client and the server the session originated from. With DNS load balancing there is no mechanism to guarantee that the client always gets connected to the same server.

2. Processing load: A DNS server is only involved during the initial part of a connection where the server IP is resolved. Once the client knows the IP, the DNS server has no natural way of getting any feedback from the traffic sent to and from the server. Thus, the best a DNS server can do is to rotate between the IP addresses in the response. There is no way to account for how demanding the client will be. In a worst-case scenario, the most demanding clients will be sent to the same server while clients generating very little activity will distributed between other servers. This will lead to an undesired imbalance where one server will be exposed to a higher load than the others.
3. Fault tolerance: A key component for any HA system is monitoring. Again, a DNS server has no natural way to determine the state of the servers behind the DNS records. In case one of the servers goes offline, a part of the clients will still be directed to the offline server. Even if the DNS-record is removed. DNS-records have a Time to Live (TTL) parameter which, even if set to low values, will cause an unacceptable delay for a HA system.

A more sophisticated solution than using DNS for load balancing is to use a Server Load Balancer (SLB). A SLB is positioned as a frontend between the client and the servers so that the backend servers only communicate with clients via the SLB. For web traffic, SLBs commonly works on either layer 4 or 7 of the OSI (Open Systems Interconnection) network model, both of which have advantages and disadvantages. With layer 4, the transport layer, less resources are required but the SLB will only be able to intercept low level connection details such as TCP-headers and port numbers to manage traffic (Sharma & Mathur, 2021). However, a layer 7 SLB can intercept the HTTP application layer and route traffic based on the actual content such as URLs, cookies, and HTTP headers (Nemeth et al., 2017). This can be useful since it for example allows for dividing static and dynamic content to different backends based on HTTP paths. In such cases, backends needing synchronization between each other have the ability to scale independently from backends with static content. Working as a reverse proxy, where the client only communicates directly with the SLB as seen in Figure 1, a HTTP SLB provides solutions the problems mentioned with DNS load balancing but it can also add additional features such as Transport Layer Security (TLS) termination. With TLS termination, the SLB will handle TLS in a Hypertext Transfer Protocol Secure (HTTPS) connection between itself and the client which in most cases eliminates the need of TLS for the web server. This means that the web servers do not need to spend resources on TLS and it also has makes it easier to manage certificates since they, in such case, only need to be installed on the SLB.

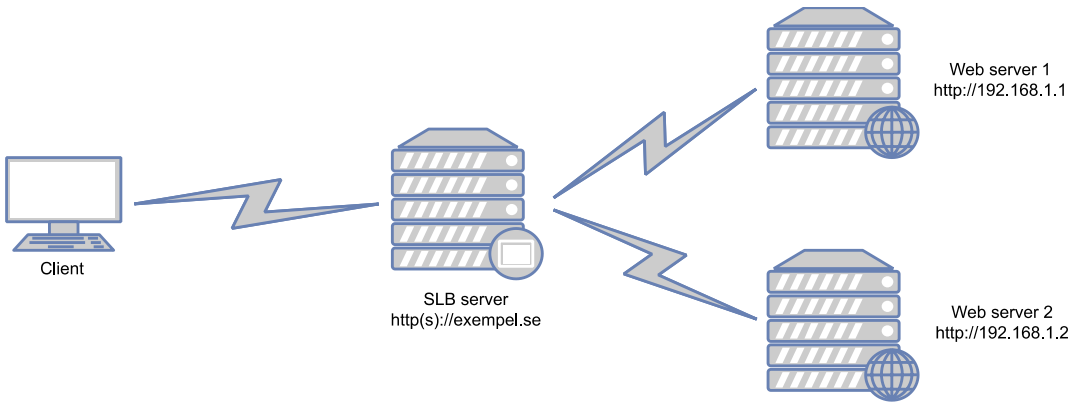


**Figure 1: Example http load balancing topology**

Many load balancers support several different algorithms to distribute traffic. These can be divided into static and dynamic algorithms (Deepa & Cheelu, 2017). In summary, static algorithms make decisions based upon compiled prior knowledge about the system. A dynamic algorithm, on the other hand, make use of current state information. Complex dynamic algorithms may take into account several policies and dynamically changing states of nodes. Nemeth et al. (2017) lists some of the overall most common algorithms which, in their simplest implementations, are static:

- Round Robin (RR): This algorithm is both very simple and commonly used. It lets servers take turn of incoming traffic in a fixed rotation order. Variations of RR exists, with static-RR being the simplest by using a preconfigured static rotation order. Weighted RR is a variant where a weight is assigned to backed nodes, either statically or dynamically, so that backends with more resources can be assigned a larger part of the load.
- Load equalization: Also known as leastconn. Here, the load balancer distributes the load to the backend node it currently has the least number of connections established with.
- Partitioning: Partitioning is designed to always connect any given client to the same server. This can be achieved by distribute a client to a server based upon a hash value of the client's IP address.

There are also two approaches, hardware-based or software-based, when implementing a load balancing solution (Moharir et al., 2020). This study will focus on software-based solutions, mainly because they generally are not proprietary and usually offer more flexibility and scalability than hardware-based solutions (Moharir et al., 2020).

## 2.3 Software Load Balancers

In this section, background of software implementing HTTP load balancing will be presented.

**HAProxy** is a popular software dedicated for load balancing. The open-source version is used by many organizations (Kondis et al., 2016). It is mostly written in C and was first released in 2001. HAProxy Technologies LCC released an enterprise version, HAProxy Enterprise Edition (HAPEE), in 2013 and the company claims that HAProxy is "… the world's fastest and most widely used open source software load balancer … "(HAProxy Technologies., 2022). The software supports load balancing both on layer 4 and layer 7 and uses text-based configuration files.

**NGINX** is one of the market leaders of web server software. According to Netcraft (2022) it has 31% of the web server market. One of the many features of this open-source web server software is that it offers the feature of being a layer 4 or layer 7 load balancer. Together with HAProxy, it is one of the most common load balancers for Linux (Nemeth et al., 2017).

**Traefik** is like HAProxy, a dedicated reverse proxy load balancer. The open-source project started in 2015 with the goal to make deployment of microservices as easy as possible. Even though it does not have an as long-established reputation as HAProxy and NGINX, it has quickly become popular and currently has over 2 billion downloads from GitHub (Miller, 2020). Traefik is mostly written in Go and offers many desired features such as, layer 4 and layer 7 load balancing, TLS termination and stickiness. Traefik is configured via their so-called *providers*. Providers can be traditional text files but also integration with orchestrators for microservices including docker, Kubernetes and Amazon ECS. One reason for Traefik's fast growing popularity is that once a provider for an orchestrator is

installed, routes can be automatically and dynamically configured for each backend offering a service via the orchestrators Application Programming Interface (API).

**Envoy** is, along with Traefik, an emerging and popular load balancing tool for containerization (Shah et al., 2019). The software is mostly written in C++ and was originally developed by Lyft until it became an open-source project under the Cloud Native Computing Foundation (CNCF) in 2017 (Woods, 2017). Envoy offers similar features as Traefik including, layer 4 and 7 load balancing. Being maintained by the same foundation as Kubernetes it also offers high integration with container-based platforms.

### 2.3.1 Previous Studies

Several previous studies have been done regarding web server load balancing performance.

Ibrahim et al (2021) did a literature review on dynamic load balancing techniques summarizing findings from nineteen previous studies regarding performance of different load balancing algorithms in various load balancing environments. They found that the choice of algorithm can have a substantial impact on web server performance depending on load balancing method and test scenario. In addition, Prasetijo et al. (2016) and Mbarek & Mosorov (2018) each did experiments comparing the leastconn and round-robin algorithm implementations in HAProxy. These two studies drew slightly contradicting conclusions. The first one suggesting that round-robin is superior when it comes to resource utilization, network connections and network request. The latter study suggests that leastconn is in general superior over RR, showing less failed connections and performing better when it comes to connection rate, response time and throughput.

Konidis et al. (2016) did an experiment implementing load balancing using Software Define Network (SDN) as a load balancing method and comparing it to HAProxy from a HA perspective. The study found that using SDN as a load balancing method has advantages when it comes to health-checking, throughput and failed connections. However, the proposed SDN solution was not able to offer many of the features that a reverse proxy-based solution could by working at the HTTP application layer.

Zebari et al. (2020) did an experiment to analyze and evaluate how HAProxy compare against Microsoft's Network Load Balancing (NLB) server role during TCP SYN flood Denial of Service (DoS) attacks. They found that NLB in general was less effected by the attacks but it should be noted that NLB only works at layer 4 and lacks many of the capabilities HAProxy can offer by working at OSI layer 7.

Few studies were found comparing the performance of load balancing software working at the application layer. Pramono et al. (2018) did an experiment comparing the performance of HAProxy and NGINX. To do this, 4 different load scenarios were applied: A *load and stress test* where the throughput of requests were measured with 500 simulated users. A similar *benchmark* test where, again, request throughput was measured with 500 users using Apache Benchmark instead of Apache JMeter. A *time test* where response times were measured with 700 users. Their last load scenario was a *click test* where clicks from 500 simulated users with a 5 second delay between them for each user where applied and response times where measured. In their *benchmark* test, HAProxy performed 19% better when the default configuration with RR were used. In their *click test* scenario where they saw response times that were over 400% longer with NGINX when using RR. Less differences were found in their other two, *time test* and *load and stress test*, experiments.

# 3 Problem Description

This section describes motives for the research and provides the research question the study is aimed to answer. In order for the work to be completed on time, limitations and objectives are also clearly stated.

## 3.1 Motivation

The reliability and performance are of great importance when it comes to a web site's economic success (Arapakis et al., 2021). Another study by Bai et al. (2017) relating to web search suggests that latency is an important factor when it comes to user experience and activity on the web. Barreda-Ángeles et al. (2015) also found that higher latencies on search engines had negative unconscious physiological effects for its users.

As stated in section 2.1, the web server traffic for popular websites needs to be distributed among several servers to achieve the required availability and performance, thus needing an effective load balancing mechanism. Envoy and Traefik are designed for hassle free integration with container solutions, making them an attractive choice for load balancing container based webservices over the more traditionally well-known alternatives HAProxy and NGINX. This integration is becoming increasingly important since container orchestrators are emerging technologies providing better performance and efficiency compared to traditional hypervisor-based platforms, simultaneously simplifying packaging and distribution of software (Singh & Singh, 2016). Shah et al. (2019) points out how major tech giants are building businesses based on container-engines and orchestrators as they simplify management and integration of virtual components. The study also mentions Envoy and Traefik as popular and emerging tools supporting containerization.

Even though several studies comparing the performance of load balancing performance have been done, such as the ones by Pramono et al. (2018) and Zebari et al. (2020), none of them include the emerging load balancing software that Traefik and Envoy are (Shah et al., 2019). Pramono et al. (2018) found significant differences in performance when comparing HAProxy and NGINX. In their default configurations HAProxy could handle 19% more requests per second compared to NGINX in their benchmark test. Even greater differences were found when they measured response time in their click test with 500 users where the latency was 19ms for HAProxy and 102ms for NGINX.

These differences in performance motivates further testing including emerging challenging software that Traefik and Envoy constitutes. This study could help system administrators with the choice of load balancing software by looking at differences with respect to performance.

## 3.2 Research Question

The aim of this study is to answer the following question:

> *How does emerging and the most popular open-source HTTP load balancers compare in terms of performance under Linux using the Round-Robin algorithm and TLS-termination?*

## 3.3 Limitations

The study has been limited to not compare any other than the four load balancing software packages, HAProxy, NGINX, Traefik and Envoy in order to answer the research question. These have been chosen for two main reasons: Firstly, they are open-source, meaning they can be used with less limitations and without any associated licensing costs. Secondly, HAProxy and NGINX are well known and established being the most common load balancers for Linux (Nemeth et al., 2017) while Envoy and Traefik are two major emerging alternatives supporting integration with container based microservices (Shah et al., 2019). These four load balancers are also well documented and actively maintained.

The chosen software packages support load balancing both on layer 4 and 7 of the OSI networking model. They also differ in terms of features and how easy they are to configure. This study is not aimed to compare any abilities for load balancing on layer 4 nor will it take all features or ease of use into account. One such feature that will not be tested is HTTP/2 even though it is offered by several load balancing software packages (Envoy, 2022; HAProxy.org, 2022).

This study will not evaluate performance using any other algorithm than non-weighted Round-Robin. Round-Robin is commonly supported and the default algorithm in all load balancers being evaluated in this study. Studies by Ibrahim et al (2021), Prasetijo et al. (2016) and Mbarek & Mosorov (2018) already looked into how different load balancing algorithms compare. When it comes to other performance related settings, the software packages will be tested with the default or recommended parameters and may not consider possible performance tweaks.

Since the focus is on the performance of the load balancers, any implementation evaluated in this study will use static content on backed web servers to optimize their performance. Even though more realistic implementations may include more complex content, such implementations will not be included in this study.

When HTTP load balancers are used to expose websites on the internet or other public networks, their robustness from a security perspective is an important factor. However, this study will not be focused on security factors.

## 3.4 Objectives

Considering the motives and limitations the following objectives were formulated in order to answer the research question:

- Select a method suitable for the study's aim
- Define performance measurements and consider validity threats
- Collect data
- Evaluate validity of data
- Analyze the data to provide an answer to the research question

# 4 Methodology

This chapter is intended to further develop the first and second objective mentioned in 3.4.

Considering the methods suggested by Berndtsson et al. (2008), the most suitable method is to perform an experiment. The other proposed methods, literature analysis, interview, case study, survey and implementation were found less suitable. A literature analysis would require more previous research directed at the research question. The aim of this study regards performance, which with a well-planned experimental approach can be quantified. An interview- or survey- based study would take a less quantitative approach and would rely on finding suitable people to interview or survey. The remaining methods, case study and implementation are also not suitable. A case study would require a case were the selected load balancers could be used and implementing new software is out of scope for this study.

The experiment method is described by Wohlin et al. (2012) as a method providing high control over execution and measurements, it also allows for easy replication and is suitable for exploring relationships between cause and effect. In this study the relationships between specific software packages and performance are to be explored, making the method suitable.

Wohlin et al. (2012) also describes a five-step method for conducting an experiment which this study follows. Those steps are:

- Scoping
- Planning
- Operation
- Analysis & Interpretation
- Presentation & Package

*Scoping* and *planning* are presented in the following sections 4.1 and 4.2. The *operation* and *analysis & interpretation* are handled in chapter 5 and 6. *Presentation & Package* are represented by this report itself.

## 4.1 Scoping

Wohlin et al. (2012) provides the template below for the scoping step. The purpose of this step is to define goals and aspects before the planning- and operation- steps in order to lay a foundation for the experiment.

> *Analyze **<Object(s) of study>***
> *for the purpose of **<Purpose>***
> *with respect to their **<Quality focus>***
> *from the point of view of the **<Perspective>***
> *in the context of **<Context>**.*

Combining the template with the aim of the study yields a scope were the goal and the purpose of the study is to:

> *Analyze **the selected software products***
> *for the purpose of **evaluate differences***
> *with respect to their **performance***
> *from the point of view of **system administrators***
> *in the context of **high-load web server systems**.*

## 4.2 Planning

The purpose of the planning step is to prepare for how the experiment is conducted (Wohlin et al., 2012).

### 4.2.1 Variables

Before the experiment can be designed, both independent and dependent variables should be defined (Wohlin et al., 2012).

#### 4.2.1.1 Independent Variables

Independent variables are the ones which can be controlled and directly changed within the experiment itself and are the following:

**The load balancer software** – *HAProxy*, *NGINX*, *Traefik* and *Envoy* are tested individually so that they later can be compared against each other. Thus, each of them is an independent variable.

**Load scenario** – Pramono et al. (2018) applies 4 different load scenarios when evaluating load balancing performance which were considered in this study. Due to the limited resource and the similarity between their *load and stress test* and the *benchmark* test only one such test, where throughput of requests is measured, will be applied. Their *time test* scenario will also be excluded for similar reasons, it is similar to their *click test* and a few details regarding the software they used could not be found. To give a broader perspective on performance, each load scenario will also be tested at various *user levels*. The two load scenarios applied in this study are further described in section 4.2.3.

#### 4.2.1.2 Dependent Variables

Dependent are variables affected by the independent variables and are in this case related to performance. A tool will be used to measure *connection rates*, *response latency* and *error rates*. To help mitigating validity threats, different variables will be measured in different load scenarios.

In the first load scenario the dependent variables will be the *connection rate* i.e., the number of connections that can be established each second as well as well as the *error rate* represented by the proportion between successful and unsuccessful connections.

In the second load scenario, *response latency,* represented by the time taken between a request and response after a connection has been established, will be a dependent variable together with the *error rate*.

### 4.2.2 Instrumentation

To provide means for performing and monitor an experiment Wohlin et al. (2012) describes three categories of instruments to consider, *objects, guidelines* and *measurement tools.* The objects in this instance are the web server environment and the load balancer configuration. Guidelines will be provided by JMeter test plans as well as scripts created to run the tests. JMeter will also be used as the measurement tool and was chosen since it is a tool that can provide accurate results and is used in many previous studies (Abbas et al., 2017). It is also a free and open-source tool with the required features for the experiment.

### 4.2.3 Experiment Design

A virtual environment was chosen to form a base for the experiment using VMware ESXi as hypervisor. Five Virtual Machines (VM) are used of which three are used as webserver backends, one as load balancing server and one client. A virtual environment was chosen over using several physical machines since higher control over resources and isolated networks capable of speeds well over 1Gb/s can be provided. Considering validity threats and potential bottlenecks, a fast internal network and control over resources are important factors to consider when designing the experiment. With the intention to evaluate the performance of the load balancing software it is important that the performance of the backend webservers is as good as possible. One of the main features with a load balancing system is the ability to scale up the capacity of backend servers as required. Therefore, is desired that the load balancer, rather than the web servers, becomes the bottleneck during the experiment. Since in this case, a virtual environment with limited resources is shared for both the load balancer and the web servers, it may be necessary configure the web servers with overall more resources than the load balancing server to avoid a situation where the performance becomes saturated by the backend servers. To further reduce this risk, static content is used and is stored on internal memory, using a tmpfs file system, for better performance. The test cases in the experiment will also be conducted directly against one of the web servers for reference. Considerations were made to use a more realistic demo website based on a Content Management System (CMS) such as WordPress or Drupal but for such setup to not become a bottleneck, it would require more resources than available on a single ESXi host.

The aim with the experiment is to evaluate performance which can vary depending on the use case and load scenario. To answer the research question, the load scenarios conducted by Pramono et al. (2018), described in section 2.3.1, have been considered and two load scenarios, described below, have been designed to evaluate performance from two different perspectives. Each load scenario is

tested with many different user levels to give a broader image of how performance is affected by the number of users. Each individual test is also repeated 30 times to provide a basis for a statistical evaluation. To improve consistency between tests, ESXi's snapshot feature is also used by letting JMeter issue a command to restore the state of the servers before any load scenario begins.

One load scenario for evaluating load balancer performance, used by both Pramono et al. (2018) and Prasetijo et al. (2016) is to perform a benchmark to test how many requests per second a web server is able to handle. Prasetijo et al. (2016) uses the tool *httperf* to measure the connection rate at various request rates and Pramono et al. (2018) does similar tests by using *Apache benchmark* and *Apache JMeter*. Based on these previous experiments, the first load scenario tested in this experiment, *Scenario1,* is designed to test a high load situation where multiple users will open a website in a very short period of time. To be able to handle a high number of requests per second is an important performance factor, for example, if a link to the website is posted on another popular website, causing a high peak of simultaneously connecting clients.

For testing *Scenario1,* JMeter is configured with a test plan where a single small static page is repeatedly requested by a HTTP GET command from the load balanced service. To test multiple concurrency levels, the experiment is repeated with varying values for thread count and loop. The thread count represents the number of users connecting to the server at once and will be tested with increased values until a certain proportion off failed connections occurs. To be able to produce enough load the loop count will be calculated so that 200 000 requests are performed in each test. This number was chosen as a compromise between how long the tests take to perform and what is needed to be able to collect enough and consistent data. The ramp-up period for the thread group was set to 10 seconds based on the experiment performed by Pramono et al. (2018), this number represents the time span between the first and last user.

A second scenario, *Scenario2,* will be based on the click test performed by Pramono et al. (2018). This scenario is designed to measure the average latency when the same user performs multiple requests and will also be tested at multiple user levels. Studies by Bai et al. (2017) as well as Ángeles et al. (2015) shows that higher latencies have substantial negative impacts on the user experience. For this test to represent more realistic scenario, JMeter is configured with a test plan to load random images from a load balanced website. For this, a sample of an image data set is installed on the backend servers. These images are then requested in the test plan with a simple controller inside a random order controller, so that 5 images are requested 20 times with a delay in between for each thread. The delay between clicks was set to 5 seconds which is the same value Pramono et al. (2018) used. This way, each thread is designed to represent one user browsing a website each clicking on 20 hyperlinks requesting 5 images per click for a total of 100 request per user.

More details of the hardware and software used when applying the load scenarios are described in section 5.1.

### 4.2.4 Validity Threats

Accounting for possible validity threats is a fundamental step in any scientific experiment. A validity threat could disrupt the relationship between what is intended to be measured and what is actually measured (Berndtsson et al., 2008). It is important to consider these in the planning step so that the data collected during the experiment can be used to generalize the result and find a valid answer to the research question. Wohlin et al. (2012) divides possible validity threats into four categories and provides a checklist of threats which may or may not be relevant for all experiments.

Validity threats from each category found relevant for this experiment will be addressed in section 4.2.5.1 – 4.2.5.4

#### 4.2.4.1 Conclusion Validity

Threats falling into this category are concerning issues with drawing the correct conclusions between the independent variables and the outcome of the experiment.

***Low statistical power*** is a threat concerning how powerful the data is when it comes to reveal true patterns. To address this threat, the experiment will be reset and repeated independently 30 times. Validity of the collected data is also further evaluated after the experiment in section 5.3 in order to verify if it has enough power to draw any conclusion.

***Fishing and the error rate*** deals with how the researcher could be biased and try to "fish" for an expected result or not properly adjusting the level of significance. The outcome of such threat could lead to conclusions which are not independent from the researchers influence. This is addressed by awareness and this planning section (4.2) itself, planning the test in advance will reduce the risk of this threat.

***Reliability of measures*** regards factors which may lead to an unacceptable deviation in data between repeated tests. For example, inaccurate instrumentation may lead to a higher deviation between repeated tests which will reduce the reliability and may result in erroneous conclusions. This is mitigated by repeating each test 30 times and consider deviations afterwards before drawing any conclusions.

***Random irrelevancies in experimental setting*** are threats coming from factors outside the experimental setting. In this experiment, it could be unplanned traffic on the network or background tasks ran by the operating system. This type of threats will be mitigated by using a hypervisor to conduct the experiment in a virtualized environment. VMware ESXi will be used in this case and can provide reliable and isolated networks between systems. It also has a snapshot feature which makes it possible to reset machines to a known state after changing internal variables. This is utilized by configuring JMeter with a setup thread group invoking commands over SSH to the ESXi host in order to reset states for the server machines before a test is executed. To ensure that no background tasks are triggered due to the real time clock, cron and systemd timers are disabled on all systems. Also, repeating each test will reduce the likelihood of this type of threat.

#### 4.2.4.2 Internal Validity

This type of threats concerns casual relations where another factor than the ones investigated may also have an effect on the result. These are categorized by Wohlin et al. (2012) into *Single group threats*, *multiple group threats* and *social threats*.

***History*** is threat in the single group which applies to this experiment since there is a risk that the history from a previous test case affects the experimental results. During this experiment this threat could be related to hardware temperature. If the ambient temperature changes or if load from previous tests have a significant impact on the system temperature, it may impact the performance of aftercoming tests. This threat is mitigated by running each test multiple times and validate the distribution and deviation between equal tests. If found necessary, it could also be mitigated by running tests before the actual test script is ran to make sure the temperature has stabilized or by adding a pause between tests to let the hardware cool down.

This threat could also include history factors related to the state of the VMs. For example, background processes could be launched at certain times by the operating system while the experiment is running or data can be cached at unexpected places. These types of threats also can affect the *Reliability of measures* and are mitigated by disabling systemd timers and the cron service as well as repeating the tests after resetting the VMs states from snapshots.

#### 4.2.4.3 Construct Validity

If construct validity threats are not concerned, an experiment may leave to much room for interpretation relating to design factors or social factors.

***Inadequate preoperational explication of constructs.*** This threat exists if constructs are not properly defined. To avoid such unclarity, measurements and inspection methods need to be well described.

***Mono-method bias*** occurs when an observation is biased because only a single type of measurement is conducted. Even though JMeter will be the only test tool used, this threat is reduced by measuring different dependent variables in the different load scenarios.

#### 4.2.4.4 External Validity

The value of the outcome of an experiment will be higher if the results can be generalized into industrial practice, which is why it is important to consider external validity threats.

***Interaction of setting and treatment*** is the main external threat applicable to this experiment. If, in this case, the software packages, would be tested in a setting far from a realistic implementation, the results would have less value. Considerations have been made to address this threat. For example, since HTTPS is used in a clear and growing majority of websites (Mozilla, 2022) TLS termination will be implemented in the experiment.

Since this experiment is conducted in a lab environment with artificial load balancing scenarios, there is a remaining risk that the load scenarios applied does not show results perfectly aligning with a real implementation and with real users. This risk is mitigated by carefully considering the test plans and by testing more than one load scenario.

While the usage of a hypervisor helps reducing threats concerning reliability of measures, random irrelevancies in experimental setting and history, it also adds an unknown factor to the interaction of setting and treatment. There is a risk that the hypervisor may have performance impacts on the server VMs that does not correspond to how they would have performed in production environment or a bare-metal installation. However, it could be argued that the evaluated load balancers are likely to be deployed in a virtual environment since one of their main features is to offer scalability which goes hand in hand with virtualization techniques. VMware ESXi is a widely used product proven to be a reliable solution for cloud and cluster infrastructure compared to other hypervisors (Bakhshayeshi et al., 2014). Considering these advantages and disadvantages ESXi is seen as a good compromise from a validity perspective. One way to further reduce this threat could be to repeat the experiment on other platforms but this would also require more resources.

There is also risk, when limiting the experiment to configurations based on the default settings, that the configurations used in the experiment does not represent what would have been used in a production environment. There is a possibility that performance could be changed by tweaking settings. A similar threat is if there is an error in the configuration so that the software is not configured as intended. To minimize this threat, the documentations for configuring the load balancers are carefully studied before the experiment. The configurations are also tested and logs are checked to verify that the software packages are configured and is working as intended. One example of this is that a “catch all” server block was added to the NGINX configuration to ensure that the server’s name is matched as done by the other software packages, which does not seem to be the case when only one server block is used.

# 5 Experiment Operation

This chapter represents the operation step in the method by Wohlin et al. (2012) and is intended to further develop the data collection- and the data validation- objective mentioned in 3.4.

## 5.1 Preparation

The latest stable version of ESXi (7.0U3) was installed on the test system, an HP EliteDesk 800 G2 with a 4-core Intel i5-6600 Central Processing Unit (CPU) and 32GB of internal memory. New port groups and virtual switches were configured to provide isolated test networks for the virtual machines. The deployed network topology is shown in Figure 2.

Five VMs with the latest stable version of Debian 11 was deployed, three to serve as web server backends, one for the load balancing software packages and the last as a client to generate the load. Each server VM was configured with 4 CPUs and 4GB of internal memory. The client was also configured with 4 CPUs but with 8GB of internal memory. On all machines, cron and all systemd timers were stopped and disabled to ensure that no background tasks would be triggered during testing. The openssh-server package was enabled and configured with public key authentication on all servers to allow for remote commands.

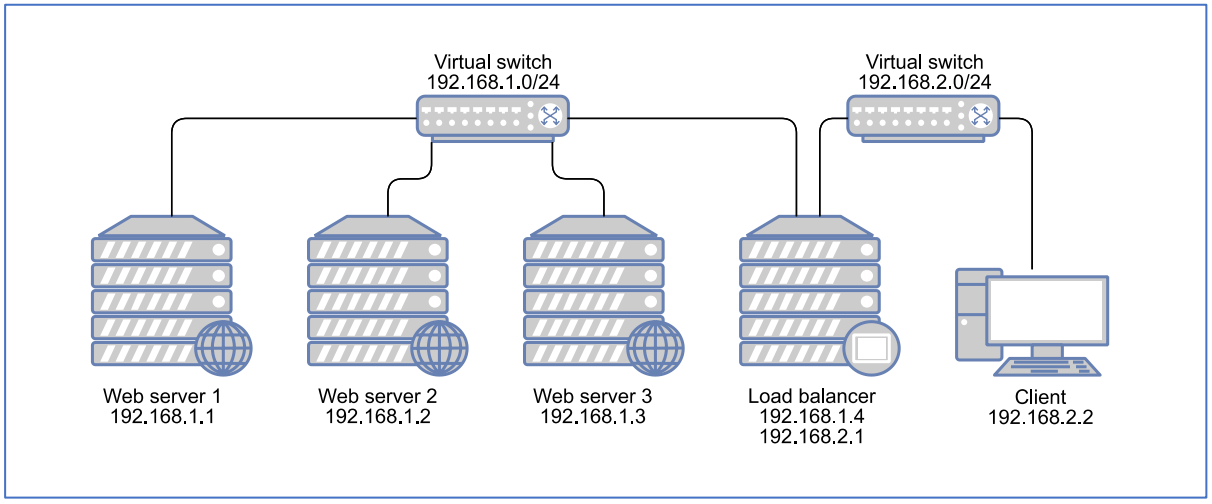


**Figure 2: Topology over virtualized environment**

### 5.1.1 Web Server Installation

Apache version 2.4.52 was installed from Debian's repository on all web server VMs. Apache is together with NGINX one of the market leaders when it comes to web server software (Netcraft, 2022) and was chosen over NGINX to avoid any bias since the NGINX load balancing feature is to be evaluated. A site was enabled with the webroot at a tmpfs drive were the static content, a 1.1kB index.html and 100 images from a flickr dataset, was installed on all servers. All web servers were configured identically except for a text entry in the index.html file indicating the server's name.

### 5.1.2 Load Balancer Installation

On the load balancer VM, the latest stable release of the four load balancers was installed. HAProxy 2.5.5 was downloaded from haproxy.org and Traefik release 2.6.3 was downloaded from their official GitHub repository. NGINX and Envoy both provide their own Debian repositories which were added to the system's package manager. NGINX 1.20.2 and Envoy 1.18.2 was then installed from their

respective repositories. A systemd unit was manually created for HAProxy, Traefik and Envoy so that systemctl could be used for starting and stopping all load balancing software packages. The intention of this is to pass a parameter to JMeter which then starts the corresponding software by invoking systemctl over SSH. All load balancers were configured with a self-signed certificate for TLS offloading and to match the hostname *lb.test* with the three backend webservers using Round Robin. The exact configurations for all load balancers are included in Appendix A.

### 5.1.3 Client Installation

On the client VM, JMeter 5.4.3 binaries were downloaded from Apache's official website and installed together with the Java package *openjdk-8*-jre offered from Debians's repository. The *hosts* file was modified with an entry to ensure that the domain *lb.test* is associated with the load balancing server. A desktop environment and Firefox was also installed to verify that all load balancers are working as intended, including verifying that all backends are used by checking for a changing server name in the html, checking that the domain name is exclusively matched and that the certificate is loaded.

### 5.1.4 Verifying Installations

After preparing the servers and the client, tests from both load scenarios were executed to ensure the setup works as intended. During these tests RAM and CPU resources were monitored on all machines with the help of *top* to make sure that the web servers are configured with enough resources to not be a bottleneck. Port utilization were checked with *netstat* to verify that no port exhaustion occurs. Logs from the load balancers were also checked to make sure no errors were logged due to misconfiguration.

## 5.2 Execution

Considering best practiced from the test tool's developer (Apache Software Foundation, 2022), a bash script was created for each load scenario to run JMeter in CLI mode. For each individual load scenario and user level the script invokes the following commands inside a loop repeating them 30 times:

```
jmeter -n -t 1.jmx -l ha-$thre-$(printf "%02d" $n).jtl -Jthreads=$thre -Jloops=$loop -Jslb=haproxy
jmeter -n -t 1.jmx -l ng-$thre-$(printf "%02d" $n).jtl -Jthreads=$thre -Jloops=$loop -Jslb=nginx
jmeter -n -t 1.jmx -l tr-$thre-$(printf "%02d" $n).jtl -Jthreads=$thre -Jloops=$loop -Jslb=traefik
jmeter -n -t 1.jmx -l en-$thre-$(printf "%02d" $n).jtl -Jthreads=$thre -Jloops=$loop -Jslb=envoy
jmeter -n -t 1-ref.jmx -l re-$thre-$(printf "%02d" $n).jtl -Jthreads=$thre -Jloops=$loop
```

Variables are used for thread- and loop- counts as well as setting which load balancer to be started after snapshots have been restored in the setup thread group. Apart from running each test against the four load balancers, they are also executed against a single web server. To achieve this, a copy of the test plan was modified to make connections directly to a web server. These test cases will be executed both with and without TLS enabled on the web server. For both scenarios, the initial thread count is set to 1 and increased by a factor of √10 until the point where the load balancers could not successfully handle a substantial part of the requests, resulting in a considerable proportion of errors. The factor of √10 was chosen since it would yield a manageable amount of user levels.

During the first scenario, no errors were seen for any of the load balancers until the user count reached 316 and at the next step, 1000 users, errors were seen from all load balancers and the experiment were stopped. The second scenario was also stopped after reaching 1000 users, at this level a substantial number of unsuccessful connections was seen from three of the load balancers.

## 5.3 Data Validation

To counteract some of the validity threats mentioned in section 4.2.4 each test case is repeated 30 times. In this section each test case is individually statistically examined to test for normal distribution and evaluate dispersion of the measured data.

Since each same exact test case is repeated, a normal distribution between the samples could give an indication of the data's quality. Outliers potentially caused by validity threats, for example, reliability of measures or random irrelevancies in experimental setting would disturb the normal distribution of the samples. With 30 samples from each test, Shapiro-Wilk Test were chosen in order to test for normal distribution and the calculated P-value is presented for each test case. This value represents the likelihood of picking the tested samples from a truly normally distributed dataset and samples are generally considered normally distributed if it is above 0.05 (Rees, 2001).

The mean, median, standard deviation, range, and coefficient of variation for each test case are also considered to further indicate the dispersion of the samples. The Coefficient of Variation (CV) is the ratio between the standard deviation and the mean (Wohlin et al., 2012) and could be useful for comparison between test cases.

Tables showing the full details of these distribution statistics for both scenarios and all test cases are presented in Appendix B.

### 5.3.1 Scenario 1

Seven different user levels were tested in the first scenario, each for all four load balancers as well as a reference test against a single web server, resulting in 35 different test cases. Data from the 30 repeated tests for each such case were parsed to calculate how many successful connections per second could be established and how many connections that were successful or unsuccessful. A script was created to parse the data efficiently and to ensure this was done correctly, samples from each test case were double checked by using JMeter's built in report generator to generate the same data.

For the five first user levels, ranging from 1 to 100 users, responses with HTTP response code 200 were received after each request consistently amongst all load balancers. In all these cases data between samples were found to be normally distributed based on Shapiro-Wilk tests. CV-values ranging between 0.37% and 1.79% which indicates a low standard deviation in relation to variances between the load balancers.

At the higher user levels, 316 and 1000, not all requests were successful. For these levels data is presented both for the amount of successful request per second and the number of errors.

With 316 users, the samples of successful requests for all load balancers were found to be normally distributed and having CV-values from 1.35% to 2.09%. Considering results from HAProxy and Envoy where a part of the responses was erroneous, the standard deviation was higher with CV-values of 191.88% and 25.23% respectively. The relatively high deviation and abnormal distribution concerning HAProxy could be explained by the fact that only a total of 35 errors was seen in all of 6 000 000 requests that the 30 repetitions include. In most of the samples, 22 of 30, all connections were successful.

At 1000 users the results between iterations of the same tests were generally less consistent than the previous user levels. The highest deviation for the rate of successful connections were seen with Traefik. In that test case the measured rate varied from 937 to 1773 resulting in a 20.39% CV. At this level Traefik also generated the most failed connections and this factor propagates to the measured rate of successful connections which may explain a higher variance.

### 5.3.2 Scenario 2

Just like in the first scenario, a script was made to parse data from 35 test cases, each conducted 30 times. This time the average latency for responses with a 200-response code is calculated and the output also contains the number of responses with any other response code. To ensure that the script works as intended samples from each test case was compared to data from JMeters report generator to ensure consistency, just as in the first scenario.

In this scenario each user makes in total 100 requests, meaning that there are fewer samples of requests in test cases with fewer users. The standard deviation was also generally higher with 1 user compared to 3 or 10 users. A similar trend can be seen after around 100 users when the deviation generally is increasing with more users. This can be seen in Figure 3 where cases with 10 and 316 users are shown with error bars representing a 95% confidence level for mean. Despite this the highest CV seen in the tests up to 316 users, where all connections were successful, was 8.90%. Shapiro-Wilks test also generates a P-value over 0.05 for all test in the range from 1 to 316 users which means that the samples between iterations are considered normally distributed.

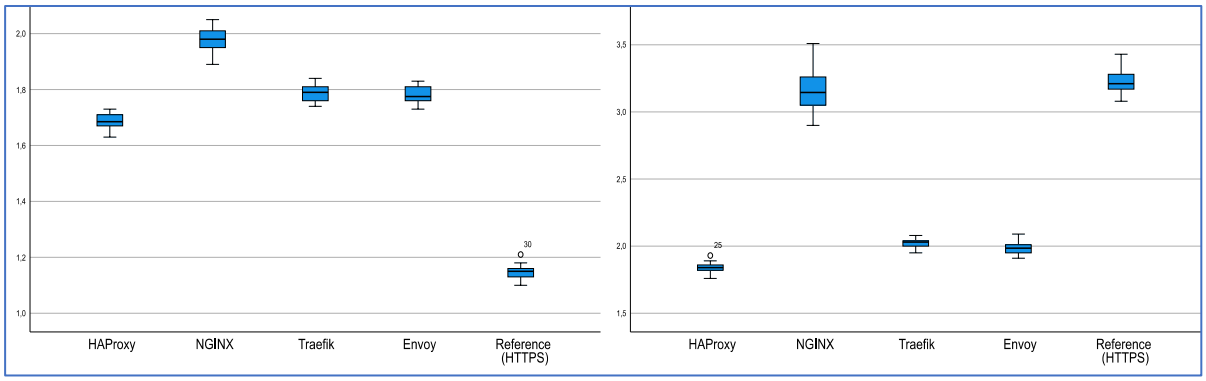


**Figure 3: Boxplot showing latencies for 10 users (left) and 316 users (right)**

Just as in the first scenario consistency is generally lower when a part of the connections fails, which started to happen with 1000 users. HAProxy was the only load balancer that did not generate any errors at this level. Still, measured response times for HAProxy varied between 5.37ms – 24.01ms which resulted in the highest CV (61.15%) amongst all load balancers in test case. This range is still relatively small considering the latency range for the closest performing load balancer was 51.64ms – 90.20ms.

# 6 Results

This chapter is a part of the *Analysis & Interpretation* step in the method by Wohlin et al. (2012). Here, data collected during the experiment are presented and will from a base for conclusions presented in chapter 7.

Considering the distribution of the data, Paired-Samples T tests were found appropriate to evaluate if the differences measured between the load balances are of significance. This test is suitable when comparing samples resulting from repeated measurements and the 0.05 level is commonly used (Wohlin et al., 2012). This means if the P-value is less than 0.05, there is also less than 5% risk that the data is considered significant even if it is not. In Appendix C, results from such T tests where all load balancers are paired against each other, including all 30 samples for each test case, are presented in tables.

For both scenarios each measuring point presented below is taken from the average value amongst 30 repeated test cases. The deviations between these cases are discussed in section 5.3 and tables including standard deviation values are included in Appendix B.

The Reference value presented in each diagram is results from when no load balancing is used. Instead, connections were made directly to the same single web server using HTTPS with the certificate installed on the web server instead of on the load balancer.

## 6.1 Scenario 1

In this scenario the time taken to get responses on 200 000 requests were measured using different thread counts to represent different numbers of users.

Figure 4 shows how many successful responses were returned by each load balancing software and by the standalone web server for reference. Any non 200 HTTP response code is not counted in this figure, even though such connections contribute to the time factor when calculating request per second.

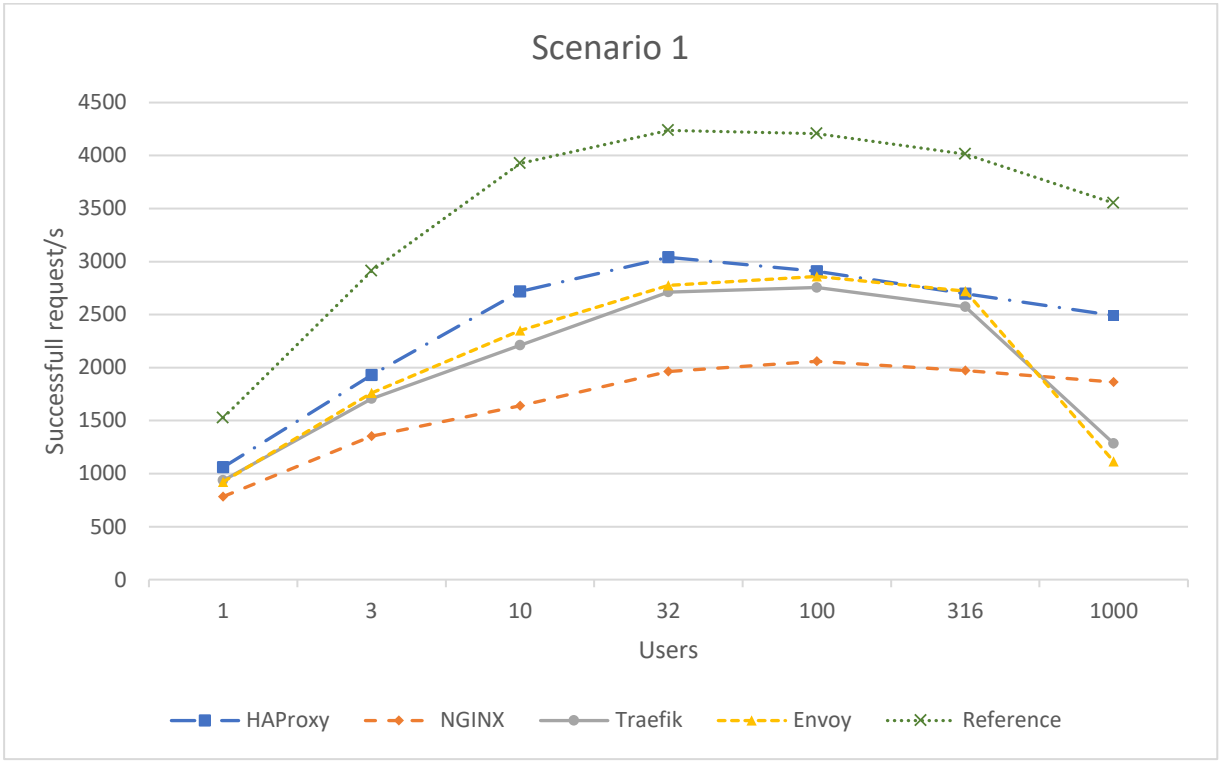


**Figure 4: Successful request per second – Scenario 1**

The differences between HAProxy, Traefik and Envoy relatively small at certain user levels. However, when paired against each other, considering all 30 samples for each test case, data from Paired-Samples T tests, found in Appendix C, shows that the differences are of statistical significance in all cases for this scenario.

Results in Figure 4 show a clear decline of performance between the two highest user levels, 316 and 1000. This coincides to some extent with the proportion of unsuccessful requests. Only for the first five user levels, ranging from 1 to 100, all requests consistently got successful responses.
When looking at responses with a non 200 HTTP response code, it was found that all such responses were reported by JMeter as having a “Non HTTP response code” indicating that no correctly formatted HTTP response was received. At 316 users failed connections were seen from HAProxy and Envoy, data from this user level is presented in Figure 5. In the final test case with 1000 users, failed connections were seen from all of the load balancers. These data are presented in Figure 6.

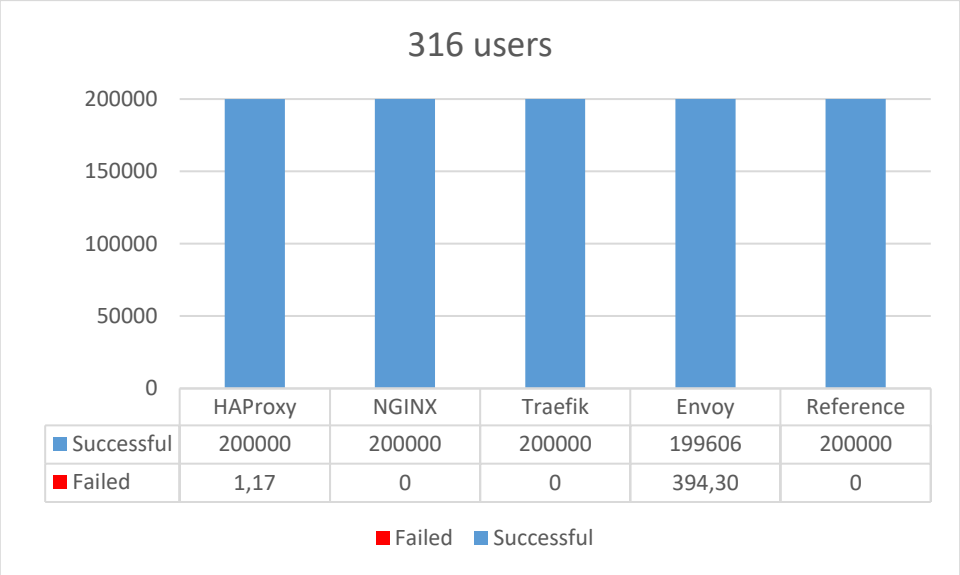


**Figure 5: Mean failed connections ratio – Scenario 1 – 316 users**

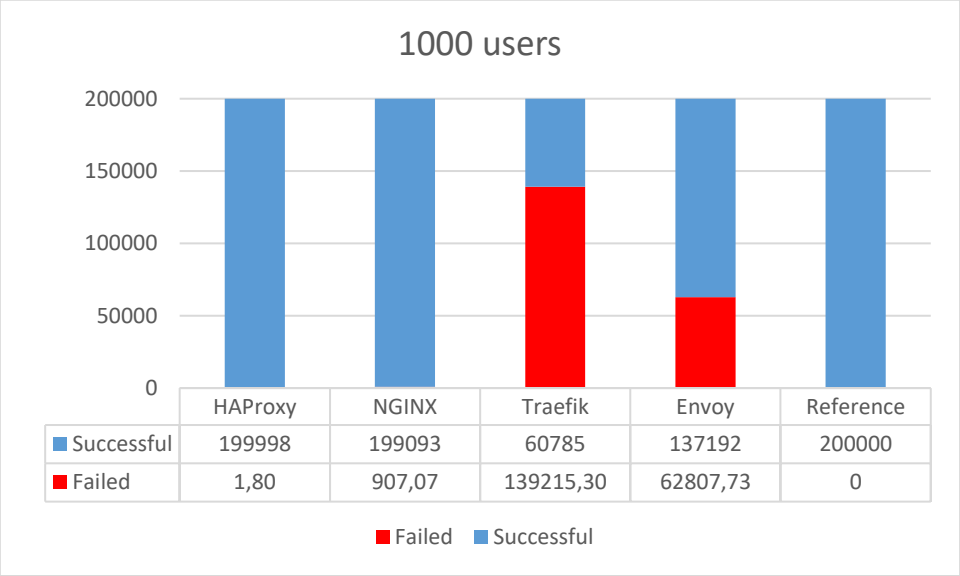


**Figure 6: Mean failed connections ratio – Scenario 1 – 1000 users**

## 6.2 Scenario 2

In this scenario each user requested 5 images at a time. This is done in random order 20 times with a delay in between. During these 20 “clicks” the average response time is measured and data is presented at 7 different user levels ranging from 1 to 1000.

Figure 7 shows the average response time for all requests at the first 6 user levels. At all these user levels all requests consistently got a response with a 200 HTTP response code. Thus, data from all requests are included.

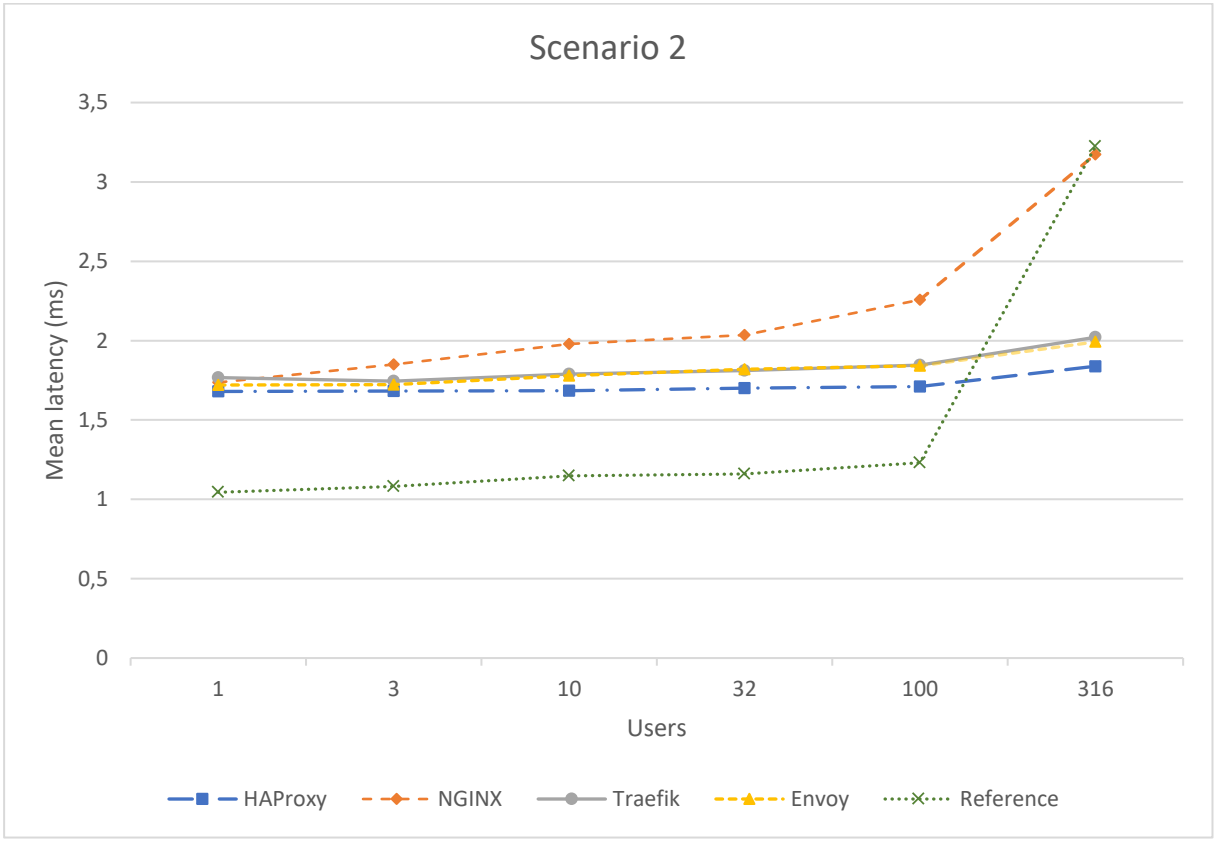


**Figure 7: Mean response latency – Scenario 2**

Just like in the first scenario differences, especially between Traefik and Envoy, are relatively small at certain user levels. Paired T tests shows that the differences at the levels with 10 and 100 users are small enough to be of statistical significance.

With 1000 simultaneous users, failed requests are seen from all load balancers except HAProxy. Just as in the first scenario, JMeter reported a “Non HTTP response code” for all responses that were not OK. Statistics showing the ratio between failed and successful responses are presented in Figure 8. Due to a substantial part of failed connections and response times several orders of a magnitude higher compared to the previous 316 user level, data for the last user level is presented in a separate Figure 9.

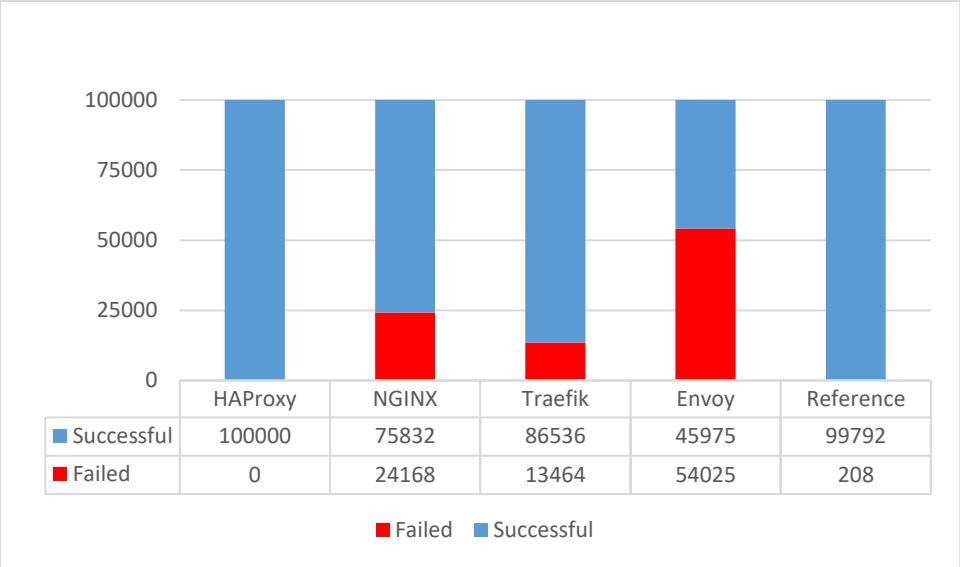


**Figure 8: Mean failed connections ratio – Scenario 2 – 1000 users**

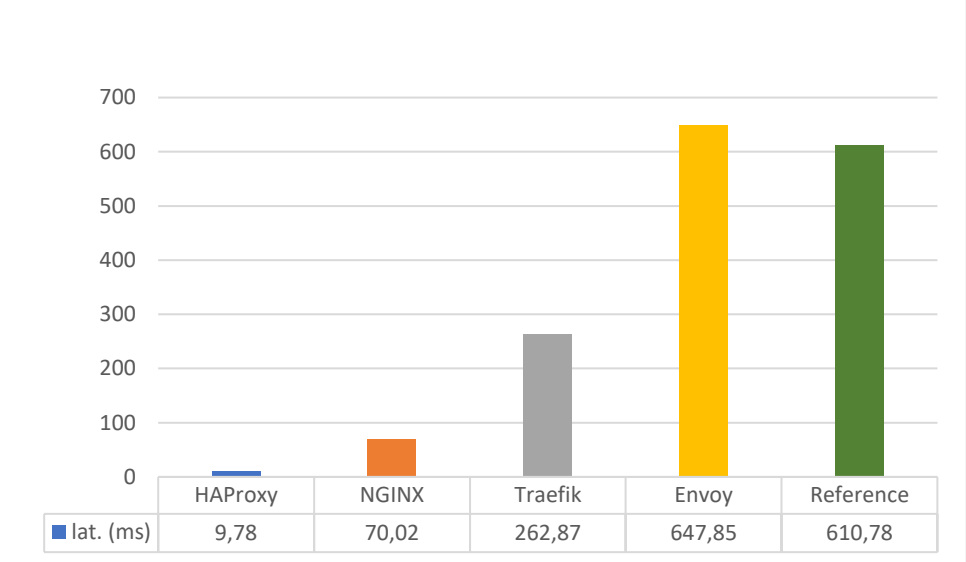


**Figure 9: Mean response latency for successful requests – Scenario 2 – 1000 users**

# 7 Conclusions

This study has identified a problem that there is a lack of guidance from previous research regarding how the emerging load balancers Traefik and Envoy perform in relation to the more well-known alternatives HAProxy and NGINX. This led to the research question this study aims to answer:

> *"How does emerging and the most popular open-source HTTP load balancers compare in terms of performance under Linux using the Round-Robin algorithm and TLS-termination"*

To answer this question an experiment was developed to test the performance with a varying number of users in two different scenarios. Taking the consistency between multiple iterations and results presented in chapter 6 in to account, both load scenarios of the experiment could reveal clear performance differences between the load balancers. Based on these results conclusions can be drawn to answer the research question.

First of all, it can be concluded that the results of this study to a large extent coincides with results from by the experiment by Pramono et al. (2018) when HAProxy and NGINX is compared. In their benchmark where 500 users were simulated, they found that HAProxy could handle 19% more requests per second. In this study similar variables were measured in the first scenario with the number of users ranging from 1 – 1000. It was found that HAProxy performed 37% better with 316 users and 34% better with 1000 users. Even higher differences seen at some of the other user levels, the highest being with 10 users where HAProxy performed 66% better. Results from Pramono et al. (2018) also showed that NGINX had over 400% longer response times at 101.89ms compared to HAProxy's 19.45ms in their click test where 500 users were simulated. This study tested similar variables in the second scenario. Considering the user levels next to 500 that were tested in this experiment, response times for NGINX were measured to be 73% longer with 316 users and 616% longer with 1000 users.

It can also be concluded that in both scenarios HAProxy generally performed the best having better performance than the other load balancers in most test cases. With the same reasoning, NGINX generally performed the worst in both scenarios having the lowest performance in most test cases. This leaves Traefik and Envoy's with request throughput and latencies generally somewhere in between HAProxy and NGINX.

As long as all connections could be handled successfully, Traefik and Envoy performed very similarly and for the most part not far from HAProxy. Still, Paired T tests show that there measured differences, with the exceptions of Traefik and Envoy at the levels with 10 and 100 users, are of statistical significance. During the first scenario, Envoy performed marginally better than Traefik and even outperformed HAProxy with 316 users. However, they both stood out in at the level with 1000 users having significantly more errors and less throughput than both HAProxy and NGINX. A similar pattern was seen in the second scenario where no considerable differences could be seen between Traefik and Envoy until the level with 1000 users. This time Envoy produced more errors and also had a larger performance decline than Traefik.

Assuming all test cases are equally important, the average performance difference from both scenarios can be calculated and this data is presented in Appendix D. The data shows that Traefik performed 13.2% better, Envoy 13.8% better and HAProxy 77.2% better compared to NGINX. However, in both load scenarios less consistency and a high proportion of unsuccessful were seen in the tests with 1000 users leading to differences orders of a magnitude higher than at the other user levels largely impacting the average. When calculating the same average performance differences excluding tests with 1000 users from both scenarios excluded, Traefik performed 24.1% better, Envoy 26.9% better and HAProxy 36.0% better than NGINX.

# 8 Discussion

Using an experimental method, this study has been able to compare performance factors of four load balancers. By conducting an experiment in a controlled lab environment, the load balancers could be tested and compared on equal terms. The virtual environment could provide high speed networking between VMs and contribute to the consistency of results between repeated tests by making it possible to manipulate machine states. The results also show that the reference measurements, were load was applied to a standalone web server, yielded the best performance in all cases except at the highest user levels in scenario 2. This shows that even though the load in that case was not distributed amongst two other servers, the backend servers, as intended, had enough capacity to not become a bottleneck in these cases. While factors of the hypervisor based virtual environment could minimalize validity threats regarding statistical power and reliability of measures, one shortcoming of this study is that there is an uncertainty of how the results can be generalized to other environments.

Even though validity threats regarding Interaction of setting and treatment were considered, a few aspects of this threat remain. First, even though more than one load scenario was tested, it can always be argued that users simulated in a lab environment never truly can represent real users. Another aspect of this is the configuration of the load balancers where this study was limited to default parameters. Solving these limitations would be a complex task that would require more resources and further studying.

Ethical aspects are important to consider in any experiment including human subjects (Wohlin et al., 2012). In this study the subjects are software on a computer system. However, it could be argued that stakeholders, developers or other passive participants involved the various software indirectly could be affected by the outcome of the study. Still, it is believed to be unlikely that any part of these open-source projects would be negatively impacted from this study but the importance of clarity, truth and other research ethics factors are still emphasized. From a societal perspective, this study can facilitate the work for people involved with web server operations but also improve the quality of services that are offered on the internet and used by a wider audience. This can be linked to goal 9.c in the United Nations (UN) goals for sustainable development aiming to “Significantly increase access to information and communications technology…” (United Nations, 2022a). A software product performing better provided the same resources could also be considered more resource efficient which is important for achieving another of the UN goals for sustainable development, goal 12.a, which aims to “Support developing countries to strengthen their scientific and technological capacity to move towards more sustainable patterns of consumption and production” (United Nations, 2022b).

# 9 Future work

This study has been focused on the performance of four load balancer software products in a specific environment and with specific load cases taking several limitations into account. Given more resources, it would be interesting to look into other aspects of performance. It would for example especially interesting to further compare how performance is affected if the HTTP/2 standard is implemented since it is expected to improve performance. Future studies could also go into more depth of what is causing the performance differences and how results are affected when a platform other than ESXi is used. In this experiment, a single web site was load balanced by matching the domain name in the HTTP header using otherwise default configurations. There is an opportunity to expand this type of study with other inspection methods and different configuration parameters, one example of this is that in the experiments by Pramono et al. (2018) NGINX performed better after the Keep-Alive option was enabled.

Some of the other limitations in this study could also be expanded upon in future work. It would be interesting to include more load balancing products, for example closed source products like Microsoft's Application Request Routing or other emerging software. There are also factors, other than performance, to consider when choosing load balancing software which this study did not consider. For example, studying how the load balancers differ in terms of features or ease of use would provide even more guidance in the choice of load balancer.

# References


Abbas, R., Sultan, Z., & Bhatti, S. (2017). Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege. *2017 International Conference On Communication Technologies (Comtech)*. https://doi.org/10.1109/comtech.2017.8065747

Apache Software Foundation. (2022). *Apache JMeter - User's Manual: Best Practices*. Retrieved 14 April 2022, from https://jmeter.apache.org/usermanual/best-practices.html

Arapakis, I., Park, S., & Pielot, M. (2021). Impact of Response Latency on User Behaviour in Mobile Web Search. *Proceedings Of The 2021 Conference On Human Information Interaction And Retrieval*. https://doi.org/10.1145/3406522.3446038

Bai, X., Arapakis, I., Cambazoglu, B., & Freire, A. (2017). Understanding and Leveraging the Impact of Response Latency on User Behaviour in Web Search. *ACM Transactions On Information Systems*, *36*(2), 1-42. https://doi.org/10.1145/3106372

Bakhshayeshi, R., Akbari, M., & Javan, M. (2014). Performance analysis of virtualized environments using HPC Challenge benchmark suite and Analytic Hierarchy Process. *2014 Iranian Conference On Intelligent Systems (ICIS)*. https://doi.org/10.1109/iraniancis.2014.6802585

Barreda-Ángeles, M., Arapakis, I., Bai, X., Cambazoglu, B., & Pereda-Baños, A. (2015). Unconscious Physiological Effects of Search Latency on Users and Their Click Behaviour. *Proceedings Of The 38Th International ACM SIGIR Conference On Research And Development In Information Retrieval*. https://doi.org/10.1145/2766462.2767719

Berndtsson, M., Hansson, J., Olsson, B., & Lundell, B. (2008). *Thesis Projects* (2nd ed.). London: Springer London. ISBN: 978-1-84800-008-7

de la Cruz, J., & Goyzueta, I. (2017). Design of a high availability system with HAProxy and domain name service for web services. *2017 IEEE XXIV International Conference On Electronics, Electrical Engineering And Computing (INTERCON)*. https://doi.org/10.1109/intercon.2017.8079712

Deepa, T., & Cheelu, D. (2017). A comparative study of static and dynamic load balancing algorithms in cloud computing. *2017 International Conference On Energy, Communication, Data Analytics And Soft Computing (ICECDS)*. https://doi.org/10.1109/icecds.2017.8390086

Envoy. (2022). *Envoy documentation*. Retrieved 3 June 2022, from https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/http/http_connection_management.html?highlight=http%202

HAProxy Technologies. (2022) *HAProxy Enterprise*. Retrieved 11 March 2022, from https://www.haproxy.com/products/community-vs-enterprise-edition/

HAProxy.org. (2022). *HAProxy docs – Starter guide* (Release 2.5.7). Retrieved 3 June 2022, from https://docs.haproxy.org/2.5/intro.html

Ibrahim, I., Ameen, S., Yasin, H., Omar, N., Kak, S., & Rashid, Z. et al. (2021). Web Server Performance Improvement Using Dynamic Load Balancing Techniques: A Review. *Asian Journal Of Research In Computer Science*, 47-62. https://doi.org/10.9734/ajrcos/2021/v10i130234

Internet Systems Consortium. (2022). *BIND 9 Administrator Reference Manual* (Release 9.18.0). Retrieved 8 March 2022, from https://downloads.isc.org/isc/bind9/9.18.0/doc/arm/Bv9ARM.pdf

Konidis, E., Kokkinos, P., & Varvarigos, E. (2016). Evaluating Traffic Redirection Mechanisms for High Availability Servers. *2016 IEEE Globecom Workshops (GC Wkshps)*. https://doi.org/10.1109/glocomw.2016.7848898

Mbarek, F., & Mosorov, V. (2018). Load balancing algorithms in heterogeneous web cluster. *2018 International Interdisciplinary Phd Workshop (Iiphdw)*. https://doi.org/10.1109/iiphdw.2018.8388358

Membrey, P., Hows, D., & Plugge, E. (2012). *Practical Load Balancing*. New York: Apress. https://doi.org/10.1007/978-1-4302-3681-8

Miller, R. (2020, 23 September). Five years after creating Traefik application proxy, open-source project hits 2B downloads. *Techcrunch.* Retrieved 15 March 2022, from https://techcrunch.com/2020/09/23/five-years-after-creating-traefik-application-proxy-open-source-project-hits-2b-downloads

Moharir, M., Shobha, G., Oppiliappan, A., Krishna GVL, R., Pandit, S., Akash, R., & Saxena, M. (2020). A Study and Comparison of Various Types of Load Balancers. *2020 5Th IEEE International Conference On Recent Advances And Innovations In Engineering (ICRAIE)*. https://doi.org/10.1109/icraie51050.2020.9358333

Mozilla. (2022). *Firefox Telemetry - Let's Encrypt Stats*. Letsencrypt.org. Retrieved 11 March 2022, from https://letsencrypt.org/stats/#percent-pageloads.

Nemeth, E., Snyder, G., Hein, T., Whaley, B., & Mackin, D. (2017). *UNIX and Linux system administration handbook* (5th ed.). Boston: Addison-Wesley.

Netcraft (2022, 28 February) *February 2022 Web Server Survey.* Retrieved 11 March 2022, from https://news.netcraft.com/archives/category/web-server-survey/

Pramono, L., Buwono, R., & Waskito, Y. (2018). Round-robin Algorithm in HAProxy and Nginx Load Balancing Performance Evaluation: a Review. *2018 International Seminar On Research Of Information Technology And Intelligent Systems (ISRITI)*. https://doi.org/10.1109/isriti.2018.8864455

Prasetijo, A., Widianto, E., & Hidayatullah, E. (2016). Performance comparisons of web server load balancing algorithms on HAProxy and Heartbeat. *2016 3Rd International Conference On Information Technology, Computer, And Electrical Engineering (ICITACEE)*. https://doi.org/10.1109/icitacee.2016.7892478

Rees, D. (2001). *Essential statistics* (4th ed.). New York: Chapman and Hall/CRC. https://doi.org/10.1201/9781315273174

Shah, A., Piro, G., Grieco, L., & Boggia, G. (2019). A Qualitative Cross-Comparison of Emerging Technologies for Software-Defined Systems. *2019 Sixth International Conference On Software Defined Systems (SDS)*. https://doi.org/10.1109/sds.2019.8768566

Sharma, R., & Mathur, A. (2021). *Traefik API Gateway for microservices*. Berkeley: Apress. https://doi.org/10.1007/978-1-4842-6376-1_6

Singh, S., & Singh, N. (2016). Containers & Docker: Emerging roles & future of Cloud technology. *2016 2Nd International Conference On Applied And Theoretical Computing And Communication Technology (Icatcct)*. https://doi.org/10.1109/icatcct.2016.7912109

United Nations. (2022a). *Sustainable Development Goal 9*. Department of Economic and Social Affairs. https://sdgs.un.org/goals/goal9

United Nations. (2022b). *Sustainable Development Goal 12*. Department of Economic and Social Affairs. https://sdgs.un.org/goals/goal12

Watada, J., Roy, A., Kadikar, R., Pham, H., & Xu, B. (2019). Emerging Trends, Techniques and Open Issues of Containerization: A Review. *IEEE Access*, *7*, 152443-152472. https://doi.org/10.1109/access.2019.2945930

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

Woods, N. (2017, 13 September). CNCF hosts Envoy. *CNCF Blog*. Retrieved 15 March 2022, from https://www.cncf.io/blog/2017/09/13/cncf-hosts-envoy

Zebari, R., Zeebaree, S., Sallow, A., Shukur, H., Ahmad, O., & Jacksi, K. (2020). Distributed Denial of Service Attack Mitigation using High Availability Proxy and Network Load Balancing. *2020 International Conference On Advanced Science And Engineering (ICOASE)*. https://doi.org/10.1109/icoase51841.2020.9436545

# Appendix A – Configuration files

## HAProxy

```
defaults
  timeout connect 5s
  timeout server 50s
  timeout client 50s

frontend ex
  mode http
  bind :443 ssl crt /etc/haproxy/cert/ex.pem
  acl test hdr(host) -i lb.test
  use_backend webservers if test

backend webservers
  mode http
  balance roundrobin
  server s1 192.168.1.1:80 check
  server s2 192.168.1.2:80 check
  server s3 192.168.1.3:80 check
```

**haproxy.conf**

## NGINX

```
server  {
        listen 443 default_server;
        server_name _;
        ssl_certificate /etc/nginx/cert/ex.crt;
        ssl_certificate_key /etc/nginx/cert/ex.key;
        location / {
                return 403;
                }
        }

server {
        listen 443 ssl;
        server_name lb.test;
        ssl_certificate /etc/nginx/cert/ex.crt;
        ssl_certificate_key /etc/nginx/cert/ex.key;
        location / {
                proxy_pass http://servers;
                }
        }

upstream servers {
        server 192.168.1.1:80;
        server 192.168.1.2:80;
        server 192.168.1.3:80;
        }
```

## Traefik

```
entryPoints:
  secure:
    address: :443
providers:
  file:
    filename: /etc/traefik/fileprovider.yml
    watch: true
```

**traefik.yml**

```
tls:
  stores:
    default:
      defaultCertificate:
        certFile: /etc/traefik/cert/ex.crt
        keyFile: /etc/traefik/cert/ex.key
http:
  routers:
    to-webservers:
      rule: "Host(`lb.test`)"
      tls: true
      service: webservers
  services:
    webservers:
      loadBalancer:
        servers:
          - url: http://192.168.1.1:80
          - url: http://192.168.1.2:80
          - url: http://192.168.1.3:80
```

**fileprovider.yml**

## Envoy

```
static_resources:
  listeners:
  - name: listener_0
    address:
      socket_address: { address: 0.0.0.0, port_value: 443 }
    filter_chains:
    - filters:
      - name: envoy.filters.network.http_connection_manager
        typed_config:
          "@type": type.googleapis.com/envoy.extensions.filters.network.
http_connection_manager.v3.HttpConnectionManager
          stat_prefix: ex-http
          route_config:
            name: local_route
            virtual_hosts:
            - name: local_service
              domains: ["lb.test"]
              routes:
              - match: { prefix: "/" }
                route: { cluster: webservers }
          http_filters:
          - name: envoy.filters.http.router
      transport_socket:
        name: envoy.transport_sockets.tls
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.
DownstreamTlsContext
          common_tls_context:
            tls_certificates:
            - certificate_chain: {filename: "/etc/envoy/cert/ex.crt"}
              private_key: {filename: "/etc/envoy/cert/ex.key"}
  clusters:
  - name: webservers
    connect_timeout: 5s
    type: STATIC
    lb_policy: ROUND_ROBIN
    load_assignment:
      cluster_name: webservers
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address:
                address: 192.168.1.1
                port_value: 80
        - endpoint:
            address:
              socket_address:
                address: 192.168.1.2
                port_value: 80
        - endpoint:
            address:
              socket_address:
                address: 192.168.1.3
                port_value: 80
```

**envoy.yaml**

# Appendix B – Distribution statistics

## 1 Scenario 1

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.897 | 0.057 | 0.067 | 0.135 | 0.597 | 0.293 |
| **Mean** | | 1060.07 | 783.37 | 935.40 | 922.33 | 1796.97 | 1526.57 |
| **Median** | | 1061.00 | 784.50 | 937.00 | 922.50 | 1798.50 | 1527.00 |
| **Std. Deviation** | | 5.41 | 6.52 | 7.29 | 3.97 | 11.13 | 8.12 |
| **Minimum** | | 1049 | 769 | 919 | 915 | 1776 | 1510 |
| **Maximum** | | 1071 | 793 | 947 | 929 | 1821 | 1543 |
| **CV (%)** | | 0.51 | 0.83 | 0.78 | 0.43 | 0.62 | 0.53 |

**Table A1: Successful requests per second – 1 user**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.543 | 0.243 | 0.331 | 0.421 | 0.785 | 0.307 |
| **Mean** | | 1929.20 | 1351.67 | 1705.07 | 1761.57 | 3426.23 | 2910.90 |
| **Median** | | 1927.50 | 1351.00 | 1706.50 | 1764.00 | 3427.50 | 2912.00 |
| **Std. Deviation** | | 11.16 | 5.76 | 10.62 | 20.24 | 18.03 | 13.45 |
| **Minimum** | | 1906 | 1338 | 1687 | 1723 | 3382 | 2889 |
| **Maximum** | | 1952 | 1362 | 1730 | 1799 | 3458 | 2944 |
| **CV (%)** | | 0.58 | 0.43 | 0.62 | 1.15 | 0.53 | 0.46 |

**Table A2: Successful requests per second – 3 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.122 | 0.239 | 0.089 | 0.098 | 0.929 | 0.472 |
| **Mean** | | 2717.47 | 1638.93 | 2210.83 | 2349.40 | 4690.27 | 3924.47 |
| **Median** | | 2720.00 | 1638.00 | 2213.00 | 2353.00 | 4688.50 | 3924 |
| **Std. Deviation** | | 14.64 | 6.07 | 17.53 | 17.29 | 27.19 | 29.45 |
| **Minimum** | | 2691 | 1625 | 2167 | 2307 | 4636 | 3869 |
| **Maximum** | | 2741 | 1653 | 2234 | 2377 | 4762 | 3979 |
| **CV (%)** | | 0.54 | 0.37 | 0.79 | 0.74 | 0.58 | 0.75 |

**Table A3: Successful requests per second – 10 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.766 | 0.657 | 0.493 | 0.760 | 0.103 | 0.937 |
| **Mean** | | 3040.10 | 1964.47 | 2711.53 | 2775.60 | 5046.57 | 4237.17 |
| **Median** | | 3041.50 | 1965.00 | 2709.50 | 2775.50 | 5054.00 | 4240.00 |
| **Std. Deviation** | | 37.24 | 11.00 | 15.00 | 13.63 | 33.78 | 29.16 |
| **Minimum** | | 2963 | 1942 | 2682 | 2748 | 4960 | 4174 |
| **Maximum** | | 3111 | 1990 | 2737 | 2800 | 5100 | 4301 |
| **CV (%)** | | 1.22 | 0.56 | 0.55 | 0.49 | 0.67 | 0.69 |

**Table A4: Successful requests per second – 32 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.188 | 0.504 | 0.864 | 0.301 | 0.265 | 0.228 |
| **Mean** | | 2907.63 | 2059.17 | 2755.63 | 2859.87 | 5019.27 | 4207.23 |
| **Median** | | 2910.50 | 2059.50 | 2754.00 | 2858.50 | 5031.00 | 4206.50 |
| **Std. Deviation** | | 52.10 | 20.05 | 30.43 | 35.09 | 51.57 | 38.44 |
| **Minimum** | | 2788 | 2010 | 2688 | 2787 | 4908 | 4140 |
| **Maximum** | | 3005 | 2100 | 2809 | 2919 | 5101 | 4292 |
| **CV (%)** | | 1.79 | 0.97 | 1.10 | 1.23 | 1.03 | 0.91 |

**Table A5: Successful requests per second – 100 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.873 | 0.156 | 0.781 | 0.909 | 0.414 | 0.435 |
| **Mean** | | 2695.73 | 1971.40 | 2572.00 | 2722.00 | 4985.97 | 4012.40 |
| **Median** | | 2695.00 | 1976.00 | 2571.50 | 2721.00 | 4989.00 | 4005.00 |
| **Std. Deviation** | | 45.96 | 41.24 | 34.71 | 44.12 | 44.31 | 54.97 |
| **Minimum** | | 2585 | 1845 | 2510 | 2624 | 4914 | 3915 |
| **Maximum** | | 2786 | 2061 | 2645 | 2808 | 5069 | 4116 |
| **CV (%)** | | 1.70 | 2.09 | 1.35 | 1.62 | 0.89 | 1.37 |

**Table A6: Successful requests per second – 316 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.000 | . | . | 0.167 | 0 | 0 |
| **Mean** | | 1.17 | 0.00 | 0.00 | 394.3 | . | . |
| **Median** | | 0.00 | 0.00 | 0.00 | 371.5 | 0.00 | 0.00 |
| **Std. Deviation** | | 2.25 | 0.00 | 0.00 | 99.49 | 0.00 | 0.00 |
| **Minimum** | | 0 | 0 | 0 | 216 | 0 | 0 |
| **Maximum** | | 7 | 0 | 0 | 611 | 0 | 0 |
| **CV (%)** | | 191.88 | . | . | 25.23 | 0 | 0 |

**Table A7: Unsuccessful requests – 316 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.003 | 0.051 | 0.006 | 0.556 | 0.502 | 0.995 |
| **Mean** | | 2491.07 | 1862.60 | 1284.87 | 1114.23 | 4818.20 | 3550.53 |
| **Median** | | 2503.50 | 1871.50 | 1227.50 | 1110.00 | 4820.00 | 3554 |
| **Std. Deviation** | | 93.81 | 70.47 | 261.95 | 79.61 | 25.95 | 85.22 |
| **Minimum** | | 2188 | 1686 | 937 | 969 | 4774 | 3359 |
| **Maximum** | | 2603 | 1992 | 1733 | 1328 | 4874 | 3728 |
| **CV (%)** | | 3.77 | 3.78 | 20.39 | 7.14 | 0.54 | 2.40 |

**Table A8: Successful requests per second – 1000 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.000 | 0.249 | 0.002 | 0.596 | . | . |
| **Mean** | | 1.80 | 907.07 | 139215.30 | 62807.73 | 0.00 | 0.00 |
| **Median** | | 0.00 | 909.50 | 148731.50 | 62318.00 | 0.00 | 0.00 |
| **Std. Deviation** | | 5.26 | 290.22 | 24629.73 | 5244.00 | 0.00 | 0.00 |
| **Minimum** | | 0 | 429 | 85704 | 53706 | 0 | 0 |
| **Maximum** | | 25 | 1402 | 164621 | 72677 | 0 | 0 |
| **CV (%)** | | 291.94 | 32.00 | 17.69 | 8.35 | . | . |

**Table A9: Unsuccessful requests – 1000 users**

# 2 Scenario 2

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.694 | 0.484 | 0.417 | 0.373 | 0.348 | 0.296 |
| **Mean** | | 1.68 | 1.74 | 1.77 | 1.72 | 0.48 | 1.04 |
| **Median** | | 1.68 | 1.74 | 1.77 | 1.72 | 0.49 | 1.05 |
| **Std. Deviation** | | 0.04 | 0.04 | 0.04 | 0.02 | 0.02 | 0.02 |
| **Minimum** | | 1.61 | 1.67 | 1.69 | 1.67 | 0.43 | 1.01 |
| **Maximum** | | 1.76 | 1.81 | 1.83 | 1.77 | 0.52 | 1.09 |
| **CV (%)** | | 2.22 | 2.25 | 2.02 | 1.42 | 4.35 | 1.64 |

**Table A10: Mean latency (ms) – 1 user**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.061 | 0.122 | 0.254 | 0.328 | 0.075 | 0.250 |
| **Mean** | | 1.68 | 1.85 | 1.74 | 1.72 | 0.61 | 1.08 |
| **Median** | | 1.68 | 1.86 | 1.75 | 1.72 | 0.61 | 1.09 |
| **Std. Deviation** | | 0.02 | 0.02 | 0.02 | 0.03 | 0.02 | 0.04 |
| **Minimum** | | 1.65 | 1.81 | 1.70 | 1.67 | 0.57 | 1.02 |
| **Maximum** | | 1.73 | 1.89 | 1.79 | 1.80 | 0.65 | 1.14 |
| **CV (%)** | | 1.41 | 1.08 | 1.40 | 1.93 | 2.62 | 3.24 |

**Table A11: Mean latency (ms) – 3 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.300 | 0.127 | 0.070 | 0.132 | 0.093 | 0.094 |
| **Mean** | | 1.69 | 1.98 | 1.79 | 1.78 | 0.67 | 1.15 |
| **Median** | | 1.69 | 1.98 | 1.79 | 1.78 | 0.67 | 1.15 |
| **Std. Deviation** | | 0.03 | 0.04 | 0.03 | 0.03 | 0.01 | 0.02 |
| **Minimum** | | 1.63 | 1.89 | 1.74 | 1.73 | 0.64 | 1.10 |
| **Maximum** | | 1.73 | 2.05 | 1.84 | 1.83 | 0.70 | 1.21 |
| **CV (%)** | | 1.48 | 2.01 | 1.74 | 1.44 | 2.36 | 1.88 |

**Table A12: Mean latency (ms) – 10 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.190 | 0.079 | 0.087 | 0.112 | 0.178 | 0.181 |
| **Mean** | | 1.70 | 2.00 | 1.81 | 1.82 | 0.67 | 1.16 |
| **Median** | | 1.70 | 2.04 | 1.81 | 1.82 | 0.67 | 1.16 |
| **Std. Deviation** | | 0.02 | 0.02 | 0.01 | 0.02 | 0.01 | 0.03 |
| **Minimum** | | 1.67 | 2.00 | 1.78 | 1.79 | 0.64 | 1.11 |
| **Maximum** | | 1.73 | 2.06 | 1.84 | 1.85 | 0.70 | 1.24 |
| **CV (%)** | | 0.91 | 0.84 | 0.81 | 0.87 | 2.04 | 2.93 |

**Table A13: Mean latency (ms) – 32 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.057 | 0.421 | 0.194 | 0.103 | 0.064 | 0.649 |
| **Mean** | | 1.71 | 2.26 | 1.85 | 1.84 | 0.71 | 1.23 |
| **Median** | | 1.71 | 2.25 | 1.85 | 1.84 | 0.71 | 1.23 |
| **Std. Deviation** | | 0.02 | 0.03 | 0.03 | 0.03 | 0.01 | 0.03 |
| **Minimum** | | 1.67 | 2.20 | 1.80 | 1.79 | 0.69 | 1.18 |
| **Maximum** | | 1.74 | 2.32 | 1.89 | 1.89 | 0.74 | 1.29 |
| **CV (%)** | | 1.14 | 1.41 | 1.46 | 1.61 | 1.63 | 2.18 |

**Table A14: Mean latency (ms) – 100 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.772 | 0.262 | 0.556 | 0.054 | 0.396 | 0.200 |
| **Mean** | | 1.84 | 3.17 | 2.02 | 1.99 | 1.38 | 3.22 |
| **Median** | | 1.84 | 3.15 | 2.03 | 1.96 | 1.37 | 3.21 |
| **Std. Deviation** | | 0.04 | 0.16 | 0.03 | 0.05 | 0.12 | 0.08 |
| **Minimum** | | 1.76 | 2.90 | 1.95 | 1.91 | 1.17 | 3.08 |
| **Maximum** | | 1.93 | 3.51 | 2.08 | 2.09 | 1.60 | 3.43 |
| **CV (%)** | | 1.90 | 5.02 | 1.73 | 2.49 | 8.90 | 2.44 |

**Table A15: Mean latency (ms) – 316 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | 0.000 | 0.762 | 0.320 | 0.438 | 0.635 | 0.175 |
| **Mean** | | 9.78 | 70.02 | 262.87 | 647.85 | 610.78 | 695.41 |
| **Median** | | 6.34 | 70.89 | 259.1 | 653.34 | 610.73 | 695.20 |
| **Std. Deviation** | | 5.98 | 9.99 | 40.01 | 34.15 | 1.13 | 5.34 |
| **Minimum** | | 5.37 | 51.64 | 192.89 | 590.35 | 608.31 | 684.86 |
| **Maximum** | | 24.01 | 90.20 | 336.73 | 712.58 | 613.25 | 708.99 |
| **CV (%)** | | 61.15 | 14.26 | 15.22 | 5.27 | 0.18 | 0.77 |

**Table A16: Mean latency (ms) – 1000 users**

| | | HAProxy | NGINX | Traefik | Envoy | Reference (HTTP) | Reference (HTTPS) |
|---|---|---|---|---|---|---|---|
| N | **Valid** | 30 | 30 | 30 | 30 | 30 | 30 |
| | **Missing** | 0 | 0 | 0 | 0 | 0 | 0 |
| **P (Shapiro-Wilk)** | | . | 0.426 | 0.103 | 0.190 | 0.266 | 0.837 |
| **Mean** | | 0 | 24168.43 | 13464.00 | 54024.53 | 207.57 | 378.13 |
| **Median** | | 0 | 24176 | 13335 | 53773 | 212 | 377 |
| **Std. Deviation** | | 0 | 108.64 | 2362.25 | 2131.43 | 37.27 | 13.92 |
| **Minimum** | | 0 | 23999 | 9581 | 49667 | 103 | 347 |
| **Maximum** | | 0 | 24384 | 17382 | 60420 | 275 | 403 |
| **CV (%)** | | . | 0.45 | 17.54 | 3.95 | 17.95 | 3.68 |

**Table A17: Unsuccessful requests – 1000 users**

# Appendix C – Paired-Samples T Tests

| Pair/No. of users | 1 | 3 | 10 | 32 | 100 | 316 | 1000 |
|---|---|---|---|---|---|---|---|
| **HAProxy - NGINX** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **HAProxy - Traefik** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **HAProxy - Envoy** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.028 | 0.000 |
| **NGINX - Traefik** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **NGINX - Envoy** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **Traefik - Envoy** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 |

**Table D1: Scenario 1, successful requests, 2-tailed significance values (P)**

| Pair/No. of users | 1 | 3 | 10 | 32 | 100 | 316 | 1000 |
|---|---|---|---|---|---|---|---|
| **HAProxy - NGINX** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **HAProxy - Traefik** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **HAProxy - Envoy** | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **NGINX - Traefik** | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **NGINX - Envoy** | 0.049 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| **Traefik - Envoy** | 0.000 | 0.002 | 0.148 | 0.047 | 0.563 | 0.007 | 0.000 |

**Table D2: Scenario 2, successful requests, 2-tailed significance values (P)**

# Appendix D – Mean data points

| | 1 | 3 | 10 | 32 | 100 | 316 | 1000 |
|---|---|---|---|---|---|---|---|
| **NGINX** | 783 | 1352 | 1639 | 1964 | 2059 | 1971 | 1863 |
| **HAProxy** | 1060 | 1929 | 2717 | 3040 | 2908 | 2696 | 2491 |
| **Traefik** | 935 | 1705 | 2211 | 2712 | 2756 | 2572 | 1285 |
| **Envoy** | 922 | 1762 | 2349 | 2776 | 2860 | 2722 | 1114 |
| **Reference** | 1527 | 2911 | 3924 | 4237 | 4207 | 4012 | 3551 |

**Table C1: Scenario 1, Mean requests per second 1 – 1000 users**

| | 1 | 3 | 10 | 32 | 100 | 316 | 1000 |
|---|---|---|---|---|---|---|---|
| **NGINX** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **HAProxy** | 135.32 | 142.73 | 165.81 | 154.75 | 141.20 | 136.74 | 133.74 |
| **Traefik** | 119.41 | 126.15 | 134.89 | 138.03 | 133.82 | 130.47 | 68.98 |
| **Envoy** | 117.74 | 130.33 | 143.35 | 141.29 | 138.88 | 138.07 | 59.82 |
| **Reference** | 194.87 | 215.36 | 239.45 | 215.69 | 204.32 | 203.53 | 190.62 |

**Table C2: Scenario 1, Mean requests per second in relation to NGINX**

| | 1 | 3 | 10 | 32 | 100 | 316 | 1000 |
|---|---|---|---|---|---|---|---|
| **NGINX** | 1.74 | 1.85 | 1.98 | 2.03 | 2.26 | 3.17 | 70.02 |
| **HAProxy** | 1.68 | 1.68 | 1.69 | 1.70 | 1.71 | 1.84 | 9.78 |
| **Traefik** | 1.77 | 1.74 | 1.79 | 1.81 | 1.85 | 2.02 | 262.87 |
| **Envoy** | 1.72 | 1.72 | 1.78 | 1.82 | 1.84 | 1.99 | 647.85 |
| **Reference** | 1.04 | 1.08 | 1.15 | 1.16 | 1.23 | 3.22 | 695.41 |

**Table C3: Scenario 2, Mean latencies 1 – 1000 users**

| | 1 | 3 | 10 | 32 | 100 | 316 | 1000 |
|---|---|---|---|---|---|---|---|
| **NGINX** | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **HAProxy** | 103.39 | 109.97 | 117.39 | 119.69 | 131.95 | 172.61 | 715.73 |
| **Traefik** | 100.95 | 107.37 | 111.23 | 111.82 | 122.63 | 159.32 | 10.81 |
| **Envoy** | 100.95 | 107.37 | 111.23 | 111.82 | 122.63 | 159.32 | 10.81 |
| **Reference** | 166.29 | 170.97 | 172.35 | 175.47 | 183.60 | 98.48 | 10.07 |

**Table C4: Scenario 2, Mean latencies in relation to NGINX**

| | 1 – 1000 users | 1 – 316 users |
|---|---|---|
| **NGINX** | 100.00 | 100.00 |
| **HAProxy** | 177.22 | 135.96 |
| **Traefik** | 112.56 | 124.67 |
| **Envoy** | 113.83 | 126.92 |
| **Reference** | 174.36 | 186.70 |

**Table C5: Mean values in relation to NGINX both scenarios**