Rate Limiter - CS 492 Proposal

Suman Chapai

April 20, 2023

1 Introduction

We propose building a programmable rate limiter as part of CS 492 to be completed in the fall of 2023. The rate limiter will accept incoming connections and forward requests to one or more servers if the requests get filtered through the specified filtering configuration (based on Application and IP layer). Packets corresponding to requests that don't pass through the filter will be dropped. In this document, we outline proposed functionalities and timeline.

2 Definitions

1. **Filter** and other tense forms of the verb connote whether a request is filtered through (meaning accepted) or filtered out (meaning rejected) by the rate limiter.
2. **Ursa** roughly pronounced as *woor-sa* may be used interchangeably with the noun phrase **rate limiter** and denotes the proposed rate limiter server described herein.
3. **Upstream** denotes the server(s) sitting behind the rate limiter. Envoy [Env] was taken as a reference for this terminology.

3 Functional Requirements

3.1 HTTP Filtering

1. Read HTTP headers, modify/add HTTP headers before sending request upstream.
2. Retrieve the number of of request based on given parameter for example HTTP header field or IPv4 or IPv6 address in the last x seconds where $0 < x < k$ for some $k \in \mathbb{Z}$.
3. Allow to define configuration specifications above in a configuration file.

4. Filter based on configuration.

5. Log requests and their filtering status.

4 Non-Functional Requirements

1. Construct a type-safe domain language to define Ursa configuration.

2. In addition to in-memory database that Ursa uses by default, allow users to specify a persistent database.

3. Expose API to update configuration used by Ursa.

5 Timeline

Deadline	Goal
Apr 30	Finalize language, submit proposal draft
May 30	Complete minimal Go server to receive/forward requests (in-memory DB, hardcoded configuration)
June 7	Make it possible to use yaml based configuration
June 30	Explore possibilities of domain specific language to define configuration
July 30	Complete server functionalities
August 30	Write integration tests
September 7	Support docker. Publish docker image running Ursa.
September 30	Setup load testing environment to note benchmarks
October 30	Work on domain specific language for defining configuration
November 30	Create and publish documentation

6 Technical Specifications

We plan on writing the Ursa server in *Go*. Code will be hosted on in the usraserver github.

7 Related Work

A rate limiter server is a part of API gateway of most, if not every, large company as it is one of the primary mitigation measures against DOS and DDOS attacks.

We have found two systems in the internet today that we will use as examples when building Ursa: Envoy [Env] is a proxy server written in *C++* that has tons of features including Application layer and IP layer rate limiting. Another example we will look at is Varnish [Var] that is a caching proxy whose caching behavior can be defined using a Varnish specific configuration language called

Varnish Configuration Language (VCL). We will look at VCL for reference when implementing the configuration language for Ursa.

References

[Env] Envoy. *Envoy Proxy*. URL: `https://www.envoyproxy.io/`.

[Var] Varnish. *Varnish Cache*. URL: `https://varnish-cache.org/`.