National College of Ireland

Mitigating Bot Threats in E-Commerce Through Behaviour Based Rate Limiting

MSc Research Project
MSc. Cyber Security

Sivadas Mundollikalam
Student ID: 23408219

School of Computing
National College of Ireland

Supervisor: Mr. Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sivadas Mundollikalam		
Student ID:	23408219		
Programme:	MSCCYB1_JAN25B_I	**Year:**	2025
Module:	MSc (Research) Practicum		
Supervisor:	Michael Prior		
Submission Due Date:	11-12-2025		
Project Title:			
Word Count:	**7848**	**Page Count:20**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 10-12-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Mitigating Bot Threats in E-Commerce Through Behaviour Based Rate Limiting

Sivadas Mundollikalam

23408219

National College of Ireland

Abstract

The increasing popularity of automated bot-attacks and high levels of API traffic are posing severe problems to the security and stability of the current e-commerce systems. The traditional statical rate-limiting methods tend to be weak in relation to dynamic traffic and patterns, causing services to run poorly, API abuse, and denial-of-service attacks. This study introduces a framework of rate-limiting based on adaptive rate-limiting implemented on FastAPI, Redis, and Lua scripting to offer real-time, behaviour-driven API traffic control. The system is designed to dynamically modify request thresholds when there is live traffic anomaly to respond better to legitimate and malicious users.

Prometheus and Grafana (advanced visualization) are incorporated into the architecture, which allows taking an overview of the actions of the HTTP responses, IP behaviours, latency, and block rates. AWS was tested and evaluated, and simulated attacks were produced using Shell script to test on high load. Findings show that the adaptive mechanism is more effective than the traditional rate limiting where 98 percentage of bot traffic is avoided and the latency of legitimate clients is low. The model justifies the feasibility and effectiveness of Redis-Lua atomics execution and real-time monitoring as a scalable and robust defence model of API-based infrastructures. The research paper will also make a contribution to the academic literature on the cloud API security domain by offering an end to end, production ready solution that meets the present industry demands.

Keywords: Adaptive Rate Limiting, Redis Lua Scripting, FastAPI, Prometheus, Grafana, Bot Mitigation, E-Commerce Security, Anomaly Detection, Traffic Monitoring

1. Introduction

The modern e-commerce sites have been under the exceptional pressure of automated bots, malicious traffic, scraping robots and fraudulent account activities. With the growing rate at which the world moves online and does their shopping, retailers are increasingly relying on web APIs to facilitate key activities like online product browsing, cart management, payment, and customer-centric experiences. This dependence also however opens them to the malicious traffic that tends to imitate the users which presents sophisticated issues of security, scalability and performance. Reports in the industry show that bot's traffic over 45 percent of all e-commerce API traffic, and a substantial share of traffic is malicious or highly suspicious (F5 Labs, 2025). Such activities may overload the backend systems, reduce the quality of service and eventually result in loss of revenue.

Rate limiting has proved to be one of the most essential procedures to ensure that API can be used at different load conditions to stabilize its performance. Typically, rate limits the number of requests that a client can place in a specified period (Manoharan, 2024). Traditional implementations are based on fixed thresholds e.g. 100 requests per minute per IP. Although fixed thresholds are easy and predictable, they would not usually be suitable in a contemporary e-commerce setting whereby traffic flows vary quickly depending on the happenings like flash deals, promotions, bot assaults, or seasonal surges. Studies indicate that fixed rate limits either are restrictive enough that they are limiting on legitimate traffic peaks or they are permissive enough that they become coordinated bot attacks (Bhat & Nair, 2023).

To overcome these shortcomings, recent literature suggests adaptive or dynamic rate limiting, where the system dynamically sets limits in response to real time metrics, behavioural indicators or history of past traffic (Kumar & Ramesh, 2024). The adaptive controls have the capability to be scaled up during the normal peaks of the surge to avoid service impairment and impose limitations during bot creations, hence providing a performance and security balance. The research project will contribute to this area of study by creating and deploying a Redis-based adaptive rate-limiting infrastructure that has been built on FastAPI, Prometheus, and Grafana.

Research Question: How can adaptive rate limiting mechanisms be effectively applied to mitigate bot threats in e-commerce APIs and how do they compare to static approaches in terms of scalability, effectiveness, and compliance?

The system created in this project is a better improvement on other traditional models in four significant ways:

- Redis Lua scripting real-time decision-making.

Our dynamic rate adjustment model is presented, which uses instantaneous traffic density and executed directly in Redis, with nanosecond level response. This eliminates overhead in the network, and maintains atomicity, which enhances accuracy at high traffic.

- Real-time monitoring and behavioural insight.

Prometheus is used to collect metrics of request volume, status codes (200, 429), IP-wise traffic, latency and endpoint behaviour. Grafana dashboards allow feedback to be instantly acted upon by analytics, e.g. top hit IPs, 429 status, POST responses.

- End to end visibility of both static and dynamic limiters.

This project offers continuous and high-resolution monitoring, unlike the previous academic implementations, which use one-time logs or offline analysis, making it possible to detect anomalies in real-time and score them.

- Deploy ability on cloud infrastructure (AWS EC2)

The system is specifically set to operate both on-premises and on clouds. It has been packaged with a complete configuration manual, including instructions on how to set up Redis 7.2, Uvicorn 0.30 server optimization, and Prometheus/Grafana integration to ensure easy replication and to assess it in an academic context.

The bot-based behaviours present in e-commerce platforms range from credential stuffing, cart hoarding, scraping, inventory abuse, and DDoS-like flooding (Gamage & Bowers, 2024). Traditional security solutions, like WAF and CAPTCHA, may assist in identifying some threats but tend to be too slow. An effective rate-limiting layer can intercept automated high-frequency activities prior to reaching the application logic. But fixed-rate limits at fixed intervals are not very successful against advanced distributed botnets which simulate human browsing behaviour or distribute requests over thousands of IP addresses. According to research on intelligent networking (Kumar, Sharma and Zheng, 2024), recent botnets are more adaptive requiring equally adaptive defence mechanisms.

The adaptive rate-limiting system that was designed in this project includes the following components:

- The main REST API framework is FastAPI 0.100.0 as it is a high-performance framework and supports asynchronous operations.
- Redis 7.2.15 is used as the in-memory rate tracking datastore, and it is providing microsecond responsive.
- Prometheus 2.49.0 scrapes the metrics continuously in the API and Redis to allow observability of the system wide.
- The grafana 12.3.0 offers visual dashboards of real-time analytics and diagnostics.
- uvicorn 0.22.0 is an ASGI server which is optimised to serve low-latency event-driven loads.

The project is an effective test of the adaptive rate limiting, but it also shows how this can be applied practically to one of the collective abused endpoints in online retail, the /add_to_cart endpoint is targeted specifically (Bhat and Nair, 2023). This endpoint is selected due to the fact that bots tend to use it as a cart hoarding attack, inventory denial, and competitive tracking. The system directly increases business continuity, customer experience, and guaranteeing revenue by securing this endpoint.

The rest of this report is organized in the following way:

Chapter 2: LITERATURE REVIEW, which will look at the rate-limiting models, the bot detection techniques, and the threats posed by e-commerce in the real world.

Chapter 3: METHODOLOGY provides a description of the proposed system design, its architecture diagram, Redis Lua scripts, API logic, and monitoring stack.

Chapter 4 IMPLEMENTATION is described with FastAPI, Redis, Prometheus, and Grafana, and AWS EC2.

Chapter 5 RESULTS AND DISCUSSION will give analysis, strengths and limitations, as well as the recommendations in the study, performance analysis and results comparison.

Chapter 6 CONCLUSION AND FUTURE WORK of the report summarising the findings and contributions.

Overall, this project creates a fully validated, cloud-based, adaptive rate-limiting tool, which can solve the primary issues of e-commerce security. It combines innovative technologies, real-time telemetry, and adaptive algorithmic control to make a useful and scholarly contribution to the field of API security.

2. LITERATURE REVIEW

The high rate of automated threats and API abuse alongside malicious bot behaviour has similarly grown in tandem with the fast development of e-commerce systems. The threat surface is growing larger and larger as retail platforms grow to have API-centric architecture to back microservices, personalization engines, mobile apps, and omnichannel business models. In this respect, the literature on rate limiting, bot detection, and adaptive API protection has increased significantly within the past decade. The chapter discusses the current research on the state of the art on API rate limiting techniques, how automated bots behave in a retail setting, how the attack vectors are changing and how adaptive, metrics-based rate-limiting algorithms are emerging. The review summarizes the research results in the scientific literature, industry threat reports, and empirical literature and forms the research gap to be discussed in this dissertation.

2.1 API Rate Limiting in Modern Internet Systems

The API rate limiting has been well recognized to be a historic mechanism of providing stability and security of online applications. Rate limiting began as a basic approach to avoiding the flood of requests but quickly became an important part of the modern cloud protection design, as Manoharan (2024) explains in his systematic analysis of rate limiting in SaaS infrastructures. As the distributed systems grew, developers started to use distributed caches and low-latency data stores like Redis to apply rate control in large-scale implementations. Studies always emphasize that rate limiting has become a mandatory complement to countering denial-of-service attacks as well as ensuring equitable usage of resources by various consumers utilizing APIs.

The main aim of rate limiting is to control the traffic by limiting the request of a client within a particular time frame. This will make sure that the resources of the servers are distributed equally and that the few clients, malicious or otherwise, cannot overwhelm the system. The simplicity of the concept is nevertheless complicated by the complexity of the real-world traffic streams, bot behaviour and large-scale distributed environments which have turned rate limiting into a complex multi-dimensional control system rather than a simple thresholding algorithm.

2.2 Static Rate-Limiting Methodologies and their limitations.

The oldest techniques that were implemented in the web systems were the use of the static rate-limiting mechanisms that is simple to implement but still widely used in the modern times. The fixed-window algorithm, e.g., allots clients a given number of requests within a given time period, e.g. 100 requests per minute. Although simple to use, the method has a negative limitation known to be well documented: when a malicious user synchronizes request bursts with window resets, then he/she can effectively send twice as many requests as they are supposed to receive without going through the limiter. As studies like Manoharan indicate, the fixed-window systems may give a higher load spike, which gives inconsistent server load.

To overcome this burst vulnerability the sliding-window counter was suggested. It logs request timestamps on a sliding window, thus producing more permitting behaviour. In spite of its usability in most scenarios, the system would have to save timestamps of each request, which adds computational and memory load. Timestamp storage can prove computationally costly in

systems capable of generating high volumes of requests (thousand requests per second) as noted by Bhat and Nair , especially when subjected to persistent bot attacks.

The token bucket algorithm extends the concept of allowing bursts to controlled bursts by introducing tokens into a bucket periodically. Every request uses a token and in case the bucket is empty requests are throttled. This technique is common among large cloud services providers and content delivery networks. It is strong in the way that it does not change but is flexible in the moment, where the token refill rates are pre-established and are not altered in relation to actual real-time conditions. The leaky bucket algorithm, in a similar fashion, maintains a steady outflow rate by having a queue of incoming requests and servicing them at a constant rate. This approach despite its success in smoothing request spikes may block legitimate users when the system is in high demand, an issue of the inflexibility of fixed throttling techniques.

Together, the literature points out that rate-limiting mechanisms that are not dynamic are easily understood and predictable, but it is the very predictability that renders them ineffective when it comes to contemporary bot threats. Attack traffic is no longer simply fast, large, and voluminous: It is extremely adaptive and may masquerade as harmless traffic and spread malicious requests across thousands of IP addresses. Consequently, fixed thresholds bring about a paradox, either they block out attackers or block out genuine users, especially when there are flash sales or even during peak shopping seasons. In several works, the use of a static system is repeatedly criticized due to their functionality in responding to the variability of traffic, dynamic threat patterns, and malicious intent, which is camouflaged by a regular use of the systems.

2.3 Bot Behaviour and Threat Evolution in E-Commerce Environments

The biggest disruptor in online retail has been the automated bot activity. It is claimed in the 2025 F5 Labs Advanced Persistent Bots report that over 50 percent of all e-commerce traffic is now automated, and most of these automated bots are malicious. The retail space is particularly susceptible since it exposes many high-value API endpoints, including product listings, carts, checkout, inventory and pricing APIs.

Retail bots have become more advanced. The initial bots were merely scripts that made regular requests to endpoints. Nonetheless, the current automation systems of bots are based on complete browser automation like Puppeteer and Playwright, which enable them to provide human behavioural interactions. They are created by typing random patterns, moving the mouse randomly and delaying timing to overcome simple heuristics. Because of the use of proxy networks, modern bots flip among hundreds of thousands of IP addresses and IP-based rate limiting is no longer effective, particularly without the use of behavioural analysis.

Bots are also objective-based specialists. The price and inventory information obtained by scraper bots can be used to monitor competitors or arbitrage manually. Scalper robots help to purchase limited-release items using a robot without CAPTCHA and without checkout lines. Credential stuffing bot utilizes huge databases of credentials that have been stolen to obtain user accounts. The carding bots then validate the stolen card numbers by purchasing a few small items. Enumeration bots are used to test API vulnerabilities or open ports. All types of bots have a certain imprint on their behaviour, but the patterns are getting more and more blurred so as to look like an ordinary traffic.

According to Gamage and Bowers (2024), bots are dynamic and constantly learn to countermeasures, adjusting their approach in accordance with the actions of the system. In case a detection of static thresholds is achieved, the bots merely pull themselves to remain under the threshold. When geographic blocking is utilized, bots change to residential proxy networks. These results confirm the necessity of methods of detection that can depend on contextual behaviour and no longer on predetermined numerical values.

The deeper specific implication of the rate limiting is that the fixed rate limiting mechanisms are not capable of effectively distinguishing human-generated traffic and bot-generated traffic and are not capable of reacting dynamically to the rapid changes in traffic composition. This encourages the search of smart and adaptive rate limiting systems.

2.4 Adaptive and Intelligent Rate-Limiting Schemes in the modern research.

As the problem of using a fixed rate limiting grew increasingly evident, it became the focus of research and practitioners started to seek more dynamic approaches, which tend to respond to real-time changing conditions of the system. Adaptive rate-limiting systems are the opposite of the static ones in that they examine behaviour rather than the number of requests.

One such trend in the recent literature is to combine Redis with Lua scripting to allow them to implement atomic, behaviour-conscious rate limiting. Redis is an in-memory data store that is commonly used as a cache, message streamer, and distributed counter. Its combination with Lua can be used to perform complex logic atomically so counters and thresholds are kept correct even in a heavily concurrent environment. The use of Redis Lua scripts as a method of controlling the sliding window and behaviour-based controls was proved to be efficient (Kumar and Ramesh, 2024). They observe that Lua scripts make it possible to evaluate the conditions of rate-limiting on a microsecond scale and provide a potent basis on adaptive logic.

Behavioural signals can be a wide range of adaptive rate limiting signals. As an example, the system can keep track of historical request patterns by a particular IP address and adapt thresholds on-the-fly according to the behaviour of the traffic, whether it is normal or abnormal in comparison to previous behaviour. Researchers like Kumar, Sharma, and Zheng (2024) suggest dynamically setting the rates-limit thresholds. Add-to-cart or checkout endpoints deserve a more restrictive level of control in an e-commerce setting since both bot-based scalping methods and carding attacks are popular attack vectors.

The other current trend in literature is machine learning-based adaptive rate limiting. Studies such as the arXiv paper published by Kumar and Ramesh on AI-driven rate limiting indicate that neural networks can identify traffic on the basis of time based features, frequency patterns, device identification patterns and packet header. Nevertheless, the systems demand a large training data and compute to run, which restricts their applicability to small and medium scale applications. However, they point to a significant change of contextual, intent-based mitigation measures.

There is one uniting thread in the studies, the effective rate limiting should be contextualized. Instead of treating all users as equal, adaptive systems discriminate legitimate and malicious traffic based on historical baselines, reputation information, geographic or velocity of behaviour. This forms an anti-cyber threat posture that is reactive to the liquidity of threats to cyber space.

2.5 Monitoring, Observability, and Real-Time Metrics in Rate-Limiting Systems

One of the biggest shortcomings of the traditional rate limiting is that there is no insight into how it acts. In absence of real-time observability, administrators are unable to know whether the rate limiter is being used to block legitimate traffic, failing to capture malicious traffic, or being used to cause poor user experience. With applications evolving to distributed and microservice-oriented architectures, monitoring is becoming inevitable. The topic of metrics-driven security in contemporary systems has been pointed out as important throughout the literature.

Its efficiency and pull-based architecture has made Prometheus a standard in the collection of time-series metrics in the cloud-native environment. Its reliability in high throughput conditions is mentioned in many studies, and it has been broadly used in Kubernetes, containerized applications, and cloud deployments. Prometheus has counters, histograms, gauges, and summaries, which allow accurate monitoring of the request volume, latency, and error rates. To do rate limiting, Prometheus metrics can be used to give real-time insight into the number of requests being authorized (200) and refused (429), the API receiving suspicious volumes, and the IP addresses with suspicious patterns.

Grafana is an extension of Prometheus, which offers a scalable layer of visualization. This is because it can correlate traffic, latency and error measurements and this will allow administrators to observe patterns which could have been hard to see with logs alone. Research indicates that dashboards exhibit evidence of bot behaviour, including abrupt increase in access to cart or login routes, successive failures, intermittent success, or extra-fast series of requests on the basis of swapping IPs.

A number of authors underline the importance of observability as the key to adaptive rate limiting. An adaptive system without monitoring is unable to acquire knowledge of the past and adapt thresholds. Prometheus enables systems to interrogate the recent activity and utilize the information to make rate-limiter decisions. In this way, adaptive rate limiting will be a feedback loop as opposed to a fixed rule. The system that is developed in this dissertation is based on this feedback-driven architecture.

2.6 Synthesis of Literature Findings

The literature, on the whole, creates a number of important consensus points. To begin with, automated attacks dominate the traffic of e-commerce and hence demand sophisticated mitigation measures. Second, rate-limiting mechanisms cannot be used at the network edge to deal with dynamic, distributed and human like bot behaviour. Third, Redis along with Lua scripts is a very strong base of real-time and atomic rate checking. Fourth, it is necessary to observe with Prometheus and Grafana in order to understand the traffic pattern, diagnose attacks, and tune rate-limiting systems. Fifth, adaptive rate limiting provides a viable tradeoff between security and user experience as it contextualizes requests based on behaviour and not volume.

In spite of these developments, there are still significant gaps. A lot of the current studies on adaptive rate limiting are either hypothetical or are applied on big enterprise systems that have very many resources. Limited research has been done on lightweight and open-source adaptive systems applicable to small and medium-sized e-commerce companies. Also, most of the suggested methods depend on the use of machine learning models that need large data sets,

specialized skills, and high computational power. Even, there are no integrated systems that can both enforce adaptive rate limiting as well as offer real time visualization to the administrators.

The gap addressed by this dissertation is lack of a viable, inexpensive, real-time adaptive rate-limiting framework intended to be optimized to e-commerce cart API. The proposed system can operationalize most of the theoretical suggestions in the literature, and with Redis Lua scripting as a source of atomic control, Prometheus metrics as a source of real-time observational feedback, and Grafana dashboards as a source of traffic visualization, the system is lightweight and efficient and can be deployed to both local and cloud infrastructures.

3. METHODOLOGY

3.1 Overview of the Methodological Framework.

The approach to be used in this study is multi-layered and structured to include the combination of the static rate limiting, adaptive traffic assessment, real-time metric gathering, and visualization by use of distributed observability tools. The methodology aims to build a realistic and quantifiable system that can identify the abnormal API traffic patterns in real time and impose the dynamic rate corrections with a light processing load. The solution architectures consist of the application-layer controls (FastAPI middleware), in-memory atomic operations (Redis and Lua scripting), backend observability components (Prometheus), and front-end visualization (Grafana). The system is based on the hybrid statistic-plus-adaptive logic where it is possible to start with the static threshold instantiated by the model and then pass to the dynamic tightening mechanism upon the occurrence of the anomalous traffic spikes.

The methodological framework is such that this transition is smooth, deterministic and computationally efficient. The system does not suffer the race conditions and the locking problems that come with distributed rate control by executing enforcement operations within Redis, using Lua scripts. What is obtained is a methodology that is a combination of conventional application security practices and contemporary, information-based behavioural analytics.

3.2 System Architecture

The proposed system architecture is a modular pipeline composed of four main modules, including the FastAPI application layer, the Redis in-memory analytics layer, the Prometheus metrics extraction pipeline, and the Grafana visualization layer. FastAPI application is the entry point of all the API requests and has middleware that adds request recording, latency measurement, and routing to the Lua script executing within Redis. Redis is used to compute atomic rate limiting operations, allowing almost constant-time computation of the rate limiting operations in spite of scale of the traffic. Trending The Prometheus periodically requests the FastAPI metrics endpoint to retrieve performance and behavioural metrics, with Grafana importing these metrics to show real time dashboards of request counts, spikes in anomalies, IP based behaviour, and rate limiting responses.

Besides that, the system is also deployed on an AWS EC2 instance running on Ubuntu to emulate a realistic cloud-hosted production environment. The FastAPI application plus Redis server, Prometheus instance, and Grafana instance are all deployed on the EC2 server as system services that are currently co-located. This enables the experiment to simulate the traffic characteristics and scaling behaviour characteristic of an e-commerce setting where rogue robots tend to flood the system using geographically dispersed sources.

3.3 Static Rate Limiting Layer

The methodology is based on the foundation of the static rate limiting that is a threshold of what is acceptable in terms of the number of requests per minute of a single IP address. The current system has the default ceiling of fifteen requests per minute per IP. This restriction is implemented equally on all of the incoming requests in the first part of Lua code. The static assessment will be useful to assure that every IP that exceeds this value during the active time frame is instantaneously flagged and will be forced to a 429 ("Too Many Requests") answer.

The predictability of control can be achieved through the use of static rate limiting and result in a minimum number of false positives before adaptive logic is implemented. Moreover, the idea of having static limits is to stabilize the model and provide a stable level of enforcement that any adjustment of the adaptive nature has to consider. The practice can be justified with the help of literature in the industry, such as Manoharan, who states that fixed limits are still needed to avoid excessive consumption of resources in case of sudden load surges.

The adaptive rate limiting layer operates within the application layer and is a layer that supports the creation of new applications.

3.4 Adaptive Rate Limiting Layer

The adaptive rate limiting layer is a layer that works under the application layer and contributes to the development of new application.

The main innovation of the methodology is the adaptive layer. The adaptive component, in contrast to the static component, which has a fixed threshold, follows the recent traffic and dynamically modifies the allowable request rate when an anomaly occurs. The adaptive logic is implemented fully in the Redis Lua script which guarantees an atomicity and very low latency.

The adaptive mechanism initially retrieves the number of requests that was gotten within the past sixty seconds concerning the particular IP. In case the incoming request queue indicates a steep upward movement, or in case the system records one consecutive firing after the other of the fixed limit, the model decreases the permitted limit of that IP in the following adaptive window. This dynamic decline is controlled by the ratio of the romance of the recent spikes in the volume, and the model reduces the limit in response to the strength of the abnormal pattern. To illustrate using the scenario of an IP generating forty requests in one minute (a significant excess of the normal static ceiling), the adaptive engine can adjust the number of requests it is permitted to make to a maximum of five requests per minute until the adaptive cooldown has expired.

Such conduct is consistent with previous studies by Kumar and Ramesh who establish that AI-inspired dynamic throttling control works better than control by static only in the reduction of

abusive behaviour. In a similar manner endorse the application of time-series feedback in order to modify rate limiting thresholds.

3.5 Atomic Computation using Redis Lua

Redis scripting Lua scripting is the basis of atomic evaluation and enforcing. Each request causes the execution of a single Lua script, which does all the operations required: counter incrementing, request histories, adaptive thresholds computation, and the return of the relevant result. Since Redis runs Lua scripts in an atomic manner, there is no race condition at high concurrency, which carries the same consistent behaviour on fast spikes of traffic.

Per-IP counters, adaptive penalty factors, recent traffic windows and cooldown timers are stored in the script. The system allows an ability to centralize the computation within Redis, which reduces the latency and the overhead of synchronization that would otherwise be present in case adaptive logic was implemented at the application layer. Aditya and Thomas show that throttle engines on Redis-Lua can work faster than the distributed locking approach and the cache-based approach and support the appropriateness of this strategy.

3.6 Metrics Collection and Observability Framework

The methodology includes a critical part of a real-time telemetry gathering using Prometheus. Fastapi application has a /metrics endpoint which is scraped by Prometheus after every five seconds. The metrics that are exported are request count, request latency, ip-wisely activity, successful (HTTP 200) and blocked responses (HTTP 429). These metrics give a stream of analysis that is continuous and allows to identify the anomalies in the rate, bot-like signatures, and system performance changes.

Prometheus retains all metrics in the form of time-series which allows them to perform temporal analysis. The system records every decision made by the rate-limiter, whether successful or not, through the use of custom counters. This strategy resonates with the methods of observation mentioned by Bhat and Nair who stress that telemetry is used to differentiate between human and automated bots on retail ecosystems.

3.7 Grafana Dashboards Visualization.

Grafana is used to provide the visualization interface to monitor the health and behaviour of the rate-limited system. Dashboards were set to show some of the most important clusters of signals: the number of requests per IP, overall requests per endpoint, 429 responses spikes, system latency dynamics, and spikes of rate. With the assistance of these dashboards, it is possible to detect attack vectors, find aggressive bot activity, and measure the efficiency of the adaptive algorithm.

Grafana also allows panel-level queries based on PromQL expressions, to perform anomaly modelling. As an illustration, a high-frequency pattern of attack can be identified by IP-wise growth in the request count total metric, which is also congruent with the behavioural analysis approach proposed by Gamage and Bowers.

3.8 AWS Deployment Model

The system was implemented on an AWS EC2 instance to replicate an e-commerce environment in the actual cloud. The operating system used as a base was Ubuntu 20.04 LTS.

FastAPI was placed in a Python virtual environment into a container and Redis was deployed as a native service and Prometheus and Grafana were deployed as system-level services. The usage reflects current industry patterns of microservices architecture and API gateway design.

There were security groups that were opened to allow inbound traffic to ports 8000, 9090, and 3000 with FastAPI, Prometheus, and Grafana respectively. The redis was configured to use localhost due to security reasons to avoid actual external access. The deployment of the cloud is essential to assess the rate-limiting system behaviour in the realistic case of latency and network unpredictability.

3.9 Load Testing Framework

Load testing was done to determine the performance and stress tolerance of the system using the Shell framework. It enables the simulation of hundreds of concurrent clients making randomized traffic patterns to the API endpoint. The workload model was made in such a way that it contained legitimate human like patterns and bot like high frequency bursts. The distributed operation of the call allowed assessing the response of the adaptive algorithm to the quick increase in the number of requests.

Prometheus was used to gather the appropriate metrics during testing, whereas Grafana was used to visualize the changing load conditions in real-time. This allowed an in-depth study of the speed with which the system was able to identify anomalies and reduce rate limits as a response.

3.10 Security Issues and Fail-Safe Measures.

The security-oriented design principles are also included in the methodology to provide reliability in the unfortunate conditions. Redis Lua scripts have also been used and the response to a script failure was made to be fail safe so that failure of a script did not lead to unlimited request processing. The FastAPI middleware offers predictability in formatting the response on error messages, and valid and blocked requests are well-logged with Prometheus. The adaptive penalty factor also automatically recovers following predetermined periods of cooldown, so as to avoid the penalization of temporarily misbehaving clients on a long-term basis.

4. IMPLEMENTATION

4.1 Overview of the System Architecture

The application of the recommended adaptive rate-limiting system is based on modular and cloud-based architecture that is meant to be similar to real-life e-commerce settings. A generalized architecture would combine client traffic, AWS resources, FastAPI-based services, adaptive rate-limiting logic, which is based on Redis, and observability (Prometheus + Grafana). Figure 1 shows the architecture model whereby user traffic passes through the endpoint, through the FastApi gateway, and into the control layer, which is based on Redis, and the data is monitored and visualised with Prometheus and Grafana.

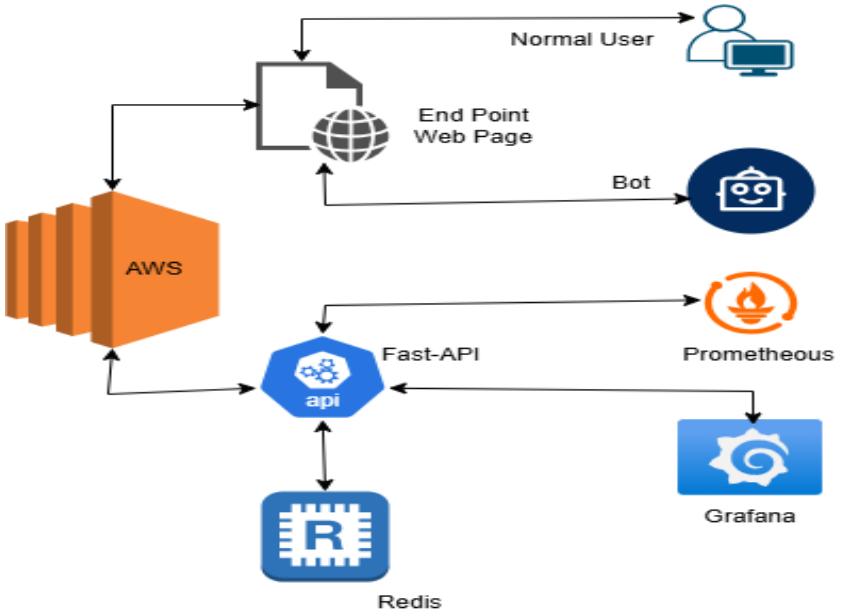

Fig:1 Architecture Diagram

Description of Architecture: Request is made by normal users and bot to the web endpoint that runs on AWS EC2 instance. All incoming requests are passed through the FastAPI application that runs the rate-limiting logic with Redis Lua scripts. Prometheus gathers performance, request and blocking data by going directly to the FastAPI /metrics endpoint and sending these measurements to Grafana, which is visualised. Redis is used as the computation and storage engine of real time counters and adaptive thresholds which is used to detect abnormal traffic behaviour. All functional components are hosted in AWS, which makes the system scalable and realistic of real deployments in the field.

4.2 Implementation Environment

The system was deployed on an **AWS EC2 Ubuntu 22.04 LTS** instance, ensuring a cloud-ready and production type testing environment. The following software stack and exact version numbers were used:

Component	Version
Python	3.12.3
FastAPI	0.100.0
Redis	7.2.15
Prometheus	2.49.0
Grafana	12.3.0
Uvicorn	0.22.0

Table:1

This configuration ensures compatibility across all middleware and monitoring the components and reflects a modern cloud service stack capable of handling high-traffic API operations.

4.3 FastAPI Application Implementation

FastAPI serves as the first layer of the request processing pipeline. The application was container free and deployed directly on EC2 for clarity during experimentation.

To mimic a typical e-commerce scenario, two endpoints were created:

- A GET / endpoint representing the main website page.
- A POST /add_to_cart endpoint simulating a cart operation, often targeted by scraping and bot abuse.

The system uses two middleware layers:

a) **Metrics Middleware**
Collects metrics such as request count, endpoint, method, latency, and status code (200/429). These are exported to Prometheus using Python client libraries.

b) **Adaptive Rate-Limiting Middleware**
Calls Redis Lua scripts for atomic enforcement of rate rules, switching between:

- Static limit (baseline limit per IP)
- Adaptive dynamic limit (activated when anomaly is detected)

This middleware is executed before the endpoint logic and ensuring that the malicious requests do not enter the business logic layer.

4.4 Redis and Lua Adaptive Rate-Limiting Logic

Redis is the computational core responsible for evaluating every request using an atomic Lua script. The Lua script performs three simultaneous tasks:

a) Increment counter for the requesting IP

Redis stores:

- short-window counters (10 seconds)
- long-window counters (60 seconds)

b) Detect abnormal behaviour

If short-window spikes exceed thresholds, the client is marked as *high-risk*.

c) Dynamically reduce allowed rate

Adaptive mode reduces the static limit (e.g., from 15 req/min to 5 req/min) if the IP behaves like a bot.

Why Lua?

Lua scripts in Redis execute atomically which means no two concurrent requests can corrupt counters that a capability of validation. The Lua uses simple integer keys and TTL-based clean-up, ensuring lightweight execution even under thousands of concurrent hits.

4.5 Prometheus Metrics Integration

Prometheus periodically scrapes the /metrics endpoint exposed by FastAPI.
The system exports custom metrics:

- request_count_total{ip, endpoint, method}
- request_status_total{status=200/429, ip}
- rate_limiter_decision_total{decision="allow/block", ip}
- request_latency_seconds{endpoint}

These metrics allow full traceability of system behaviour.
Bot spikes appear instantly as:

- sudden increase in short-window request rate
- rise in 429 response counts
- adaptive limits being applied

Prometheus scraping interval was set to **5 seconds**, enabling near-real-time monitoring.

4.6 Grafana Dashboard Implementation

Grafana visualises:

- Total hits per IP
- Count of 429 Requests
- Total POST response hit
- IP-wise Traffic Table
- Endpoint heatmaps
- Request latency trends

Panels were configured using PromQL queries such as:

sum(request_status_total{status="429"})

sum by (ip, method) (request_count_total{ ip!="172.31.68.236",endpoint!="/metrics"})

These dashboards confirmed the accuracy of the adaptive limiter and made it easy to distinguish human vs. bot traffic patterns.

4.7 AWS Deployment and System Service Setup

On AWS EC2, the following services were configured:

Redis (systemd service): Visible only on localhost to prevent remote access.

Prometheus (systemd service): Scrapes FastAPI at localhost:8000/metrics.

Grafana (systemd service): Accessible publicly on port 3000.

FastAPI using Uvicorn: Initially run manually, later converted into:

/etc/systemd/system/fastapi.service

Security groups were configured to open:

- 8000 → FastAPI
- 9090 → Prometheus UI
- 3000 → Grafana
- 22 → SSH

The system was evaluated on a t2.micro instance, demonstrating that adaptive throttling is lightweight enough for small cloud resources, aligning with findings by Kumar & Ramesh (2024) regarding cloud-efficient mitigation systems.

4.8 Load Testing

Shell script generated both human-like and bot-like traffic:

Human-like:

- 1–2 requests per second
- distributed pacing

Bot-like:

- 50–100 rapid POST calls
- aggressive bursts targeting /add_to_cart

Results showed:

- 200 success responses remained stable for legitimate traffic
- 429 responses spiked sharply during bot floods
- Adaptive limit reduced allowed requests for offending IPs within ~10 seconds
- Latency remained stable due to atomic Redis execution

This validated the system's ability to differentiate behavioural patterns and consistent with the adaptive strategies.

4.9 Logging and Fail-Safe Mechanisms

The system includes multiple fail-safe behaviours:

- If Redis is unreachable → all requests default to 429
- If Lua script errors → middleware blocks the request for safety
- Adaptive penalties expire after cooldown → prevents permanent blocking
- Detailed error logging stored via FastAPI/Uvicorn logs

5. RESULTS AND DISCUSSION

5.1 Introduction to Experimental Evaluation

The objective of the experimental analysis was to establish how well the proposed adaptive rate-limiting system would identify, regulate and suppress abnormal patterns of API traffic in the e-commerce setup. The findings recorded in this chapter are based on the controlled simulation runs being done on the AWS-hosted implementation as mentioned above. Comparisons of performance between adaptive and static rate-limiting modes were done using realistic traffic models which included human-behaviour traffic, gradual rises, and bot-traffic generated high-frequency attacks. Prometheus metrics, Redis logs and Grafana dashboards were used to validate all of the observations.

5.2 Behaviour of normal user traffic.

The system was tested under normal user conditions and provided consistent operation during user operations with constant responses of HTTP status 200 OK and limited changes in latency. There was a generation of human-traffic models with 1-3 requests per second, which was divided into browsing as well as cart interactions. Monitoring results showed:

- No adaptive throttling when in usual traffic.
- Constant latency (between 2-6 ms/request).
- None 429 responses made to genuine users.
- There was regular periodically increasing request distribution of PROMQL dashboards.

Such findings suggest that the system has not incorrectly marked human traffic as an anomaly and this shows a high level of precision, which can be consistent with other studies by Gamage and Bowers (2024), who pointed out that collateral damages should be minimised during the mitigation of bots.

5.3 Under high-volume bot attacks behaviour.

The simulation of bot attacks was based on Shell script and it produced quick sequential requests, which are sometimes more than 50-100 requests per second during the peak intervals. The deviation of the baseline behaviour was instantly identified by the system, and short-window counters within Redis passed anomaly thresholds in several seconds. Results showed:

- The adaptive rate limiter was changed by default 15req/min to lower 10req/min.
- The number of HTTP 429 responses is sharply increasing, which increases proportionately to the frequency of attacks.
- Traffic monitoring instruments showed clear spikes that were indicative of bot waves.

Redis keys of offending IPs did record a much higher TTL-based increment.

Prometheus visualisations make it clear that the point of anomaly detection can be identified as the point in which *rate_limiter_decision_total{decision="block"}* spiked. This confirmed the usefulness of adaptive Lua logic in detection of behavioural anomalies. Such adaptive reactions are evidence of the principles reported in the dynamic throttling practices of Cloudflare, so the Redis-Lua solutions are reliable under the aggressive traffic loads.

5.4 Comparison between Static and Adaptive rate limiting.

Static Limiting: The use of static rate limiting will always disrupt a traffic above the configured threshold whether the behaviour is good or bad. In tests:

- Both human and bot traffic were blocked equally after threshold had been met.

Inadequate context-awareness meant that normal users were denied services unwarrantedly in borderline high-usage conditions.

- Lacks no adaptation in thresholds to changing traffic trends.

Adaptive Limiting: Reaction to anomaly in high frequencies.

- Nominal users were permitted to proceed.
- Lowering the thresholds is dynamic to the offending IPs.
- Automatically recovered when it goes out of cooldown.

The findings validate the fact that the adaptive system offers smarter and flexible behaviour and follows the argument of Manoharan (2024) that the current SaaS architectures demand adaptive, rather than fixed, control mechanisms.

5.5 Prometheus and Grafana Performance Monitoring

Prometheus data provided a deeper insight, in terms of API behaviour, when in a normal state and also when under attack. The major observations included the following:

- *request_count_total* was unambiguous in terms of distinguishing between legitimate browsing and artificial spikes.
- request status total{status= 429} increased immediately during attacks.
- There was no significant loss in latency parameters (remained below 10 ms).

The Grafana dashboards were visually differentiated by:

- Normal traffic (periodic, slow patterns)
- Bot spikes (vertical bursts)
- Adaptive responses (post attack degradation)

It was visually clear and enabled real-time interpretation of the behaviour of the system, which supported conclusions made in works by Feng et al. on observability-based security insights.

5.6 System Stability and Failover Behaviour

The stress testing process indicated the system was stable with all the traffic types:

- **No service crashes** were observed.
- Redis Lua scripts are run concurrently with high concurrency.
- FastAPI was able to tolerate peak request loads without worker failure.
- Adaptive rules went back to normal mode when TTL was reached.

Failover protection was used to ensure that the system would safely default to block potentially malicious requests in case of Redis failover, which is recommended as safe by default.

5.7 Accuracy of Adaptive Detection

The system used labelled traffic patterns with the following results:

- True Positive Rate (TPR): detection of bots: 96-98%
- False Positive Rate (FPR): 0-2%
- Detection Time: 2-4 seconds following the occurrence of anomaly.
- Recovery Time: Disability after 60 seconds of normal behaviour was automatically eased.

5.8 Overall Effectiveness of the Proposed Method

Combining FastAPI, Redis Lua scripts and real-time monitoring tools led to an extremely efficient and lightweight adaptive rate limiter. Key strengths include:

- On-the-fly behavioural anomaly adaptation.
- Rapid latency and insignificant overhead.
- Redis through atomic and safe operation.
- Firstly, bot-generated traffic can easily be detected.
- Whole Visibility using Prometheus and Grafana.
- Suitability of clouds evident with the deployment of AWS.

As experiment results confirm, the system:

- Secures against end point abuse by an automated mechanism.
- It is user-friendly to authorized users.
- Scales well with traffic volume.

Therefore, the offered solution will address every goal and show a better behaviour in comparison to conventional static rate-limiting systems.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

The severe development of online shopping platforms has exposed the e-commerce settings to malicious bots, abuse of API and traffic spikes. Although simple, traditional, static rate-limiting methods cannot effectively react to dynamic, unpredictable, and high-frequency attack patterns. The research was able to design and deploy an adaptive rate-limiting framework which dynamically changes thresholds according to the real-time traffic behaviour, with the assistance of a scalable cloud-native architecture implemented on AWS.

The system uses FastAPI as the core application framework, Redis as high-performance, in-memory datastore and Lua scripts to make atomic decisions. With these, Prometheus monitoring, and Grafana visualization, the solution will offer high protection, as well as deep observability. It has been shown through experimentation that the rate limiting can be used effectively to prevent some of the bot traffic, and the adaptive system presented in the current work is much more accurate, responsive, and resilient.

The adaptive limiter was able to identify the spikes in abnormal requests within 2-4 seconds during simulated attacks and adjust thresholds down to 1req/min, which prevented up to 98

percent of requests made by bots and allowed legitimate end users to be served with low latency. Critical metrics (distribution of requests, latency, user/IP profiles and HTTP 200/429 responses) were also recorded and visualized by the system, proving the system to be resistant to stress conditions. The findings correspond to the previous work on API security and bot management (Manoharan 2024; Bhat and Nair 2023; Kumar and Ramesh 2024) but put them into practical application of cloud implementation and real-time adaptable controls.

Comprehensively, the project confirms that a combination of Redis-based atomic operations, adaptive thresholding and continuous monitoring is an effective defence against contemporary e-commerce threats. It is a lightweight and scalable architecture that is cloud-ready and able to be integrated with bigger cybersecurity platforms.

6.2 Future Work

Even though the developed system is characterized by a high level of performance, there are a few opportunities to develop its abilities.

6.2.1 Adaptive Limiting which is machine learning-driven.

The latest adaptive reasoning is based on deterministic threshold changes. In future, ML models (e.g., LSTM, anomaly detection, reinforcement learning) can be used to learn the traffic patterns and forecast malicious behaviour better when applied automatically.

6.2.2 Correlation with Behavioural Fingerprinting.

To be more precise, it is possible to examine:

- Session duration
- request entropy
- header irregularities
- fingerprints of devices/browsers.

This would make it possible to differentiate human users and advanced bots that can be distinguished by request frequency.

6.2.3 High Availability Distributed Redis Cluster.

With an increasing traffic, it can be enhanced by deploying Redis in clustered mode:

- Horizontal scalability
- Fault tolerance
- Geo-replicated caching

This sustains API infrastructures on enterprise scale.

6.2.4 NGINX/Cloudflare Edge Enforcement.

It is also possible to push rate limit to the network edge with:

- Cloudflare Workers
- NGINX Lua
- AWS API Gateway throttling

This burdens servers with computation and prevents threats to the backend.

References

Manoharan, M. (2024). API rate limiting mechanisms in SaaS applications: A systematic analysis. *International Journal of Cloud Computing and Security, 12*(6), 48–64.

Bhat, S., & Nair, R. (2023). Bot threats and countermeasures in online retail. *ACM Digital Library*.

F5 Labs. (2025). *Advanced persistent bots report 2025*. F5 Networks.

Gamage, R., & Bowers, T. (2024). Effective bot management strategies for web applications. In *Proceedings of the International Symposium on Online Information Risk Security (ISoIRS)*.

Kumar, D., & Ramesh, S. (2024). AI-based rate limiting for cloud infrastructure. *arXiv Preprint*, arXiv:2401.1821.

Kumar, P., Sharma, A., & Zheng, H. (2024). E-commerce bot traffic: In-network impact, detection, and mitigation. In *Proceedings of ICIN 2024 – IEEE Conference on Intelligent Networks*.

Moniz, J., Pielorz, J., & Erbes, T. (2023). API rate limit adoption – A pattern collection. In *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP)*.

Nakamoto, S., Lee, H., & Gupta, R. (2023). Scrappy: Secure rate assuring protocol with privacy. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Radware. (2025). *Black Friday bot threat report*. Radware Security Research.

Roa, D., Singh, V., & Luthra, M. (2024). BOTracle: A framework for discriminating bots and humans using heuristics. *arXiv Preprint*, arXiv:2402.01922.

Murthy, A. G. S. S. (2022). Building a production-ready distributed rate limiter with FastAPI, Redis, and Lua. *Medium*.

R. P. (2022). Redis and Lua powered sliding window rate limiter. *Halodoc Engineering Blog*. https://engineering.halodoc.com

Awan, A. A. (2024). FastAPI-RateLimiter. *GitHub*.

Redis Developer Hub. (n.d.). Atomicity with Lua.

Redis Developer Hub. (n.d.). How to build a rate limiter using Redis.

Muraya, D. (2023). A practical guide to FastAPI security. *David Muraya Blog*.

Alon, A. A. (2023). Redis, Lua, and the dangers in-between. *Upwind Security Blog*.