Analysis of HTTP DDoS Flood Attacks on Apache2 and Nginx Web Servers with Linux Ubuntu

Conference Paper · July 2024

DOI: 10.1109/ISITIA63062.2024.10668064

3 authors, including:

Haotian Tang
UNITEC Institute of Technology
2 PUBLICATIONS **1** CITATION
SEE PROFILE

Samad Kolahi
UNITEC Institute of Technology
60 PUBLICATIONS **431** CITATIONS
SEE PROFILE

Analysis of HTTP DDoS Flood Attacks on Apache2 and Nginx Web Servers with Linux Ubuntu

Haotian Tang
School of Computing
Unitec Institute of Technology
Auckland, New Zealand
tangh19@myunitec.ac.nz

Samad Salehi Kolahi
School of Computing
Unitec Institute of Technology
Auckland, New Zealand
skolahi@unitec.ac.nz

Liam Thompson
Event Hospitality and Entertainment Industry
Auckland, New Zealand
liam@crypt.gen.nz

***Abstract*—With the development of the Internet, HTTP Flood attacks, a method of cyber-attack that floods web servers with ostensibly authentic network Requests and causes them to lose their ability to provide normal services, are increasingly favoured by attackers. This paper investigated and evaluated the ability of the most recent versions of Apache and Nginx web servers to handle legitimate Requests during the HTTP DDoS Flood attack on Linux Ubuntu (22.04) in a controlled, real testbed environment. Two metrics, Transactions Per Second (TPS) and HTTP Requests Error Rate were collected and analysed. In the experiment, the Apache web server was attacked by malicious HTTP Request traffic of 10,000 Threads, each generating 10,000 Requests, Apache remained overloaded from the 48th second to the end and could not respond to the legitimate user. In contrast, Nginx was not always overloaded. Our experimental results show that Nginx can better handle legitimate Requests during HTTP DDoS Flood attacks due to its better Concurrency design and Event-Driven, Asynchronous Non-Blocking architecture.**

Keywords—HTTP Flood Attack, DDoS Attack, Apache, Nginx, Web Server, Performance Analysis

I. INTRODUCTION

The growth of the Internet has transformed various aspects of people's lives, including communication, banking, work, education, entertainment, shopping, and healthcare, among others. This transformation is evident in the emergence of social apps, remote meetings, telemedicine, online shopping, and online gaming. However, along with these advancements, security risks and vulnerabilities are associated with these network-related infrastructures.

One of the most prevalent threats among numerous cyber security issues is Distributed Denial of Service (DDoS) attacks. In DDoS attacks, attackers control many zombie hosts, and send a large volume of malicious traffic to consume server resources and network bandwidth. DDoS attacks cause servers to not provide normal network service for legitimate users [1].

In recent years, hackers have preferred Layer 7 (application layer) DDoS attacks [2] over other types. HTTP DDoS attacks are the most common type of Layer 7 attacks. In the third quarter of 2023, Cloudflare counted 8.9 trillion HTTP DDoS Requests, an increase of 65% compared to 5.4 trillion in the second quarter of 2023 [3]. Hackers favour HTTP DDoS Flood attacks is that this attack is different from classic DDoS attacks (e.g., the UDP Flood attack [4, 5], the Surmf attack [6], and the TCP SYN Flood attack [7]), the HTTP DDoS Flood attack does not rely on faked packets to conceal the attacker's real IPs. When conducting an HTTP DDoS Flood attack, hackers manipulate the botnet to make HTTP Requests, forcing the web server up to the amount of simultaneous Requests limit. The manipulated IP addresses are obtained from legitimate users and dispersed across multiple areas. As a result, these attack Requests are indistinguishable from regular users browsing websites and can baffle network specialists. This aspect makes combating HTTP DDoS Flood attacks more challenging than traditional DDoS [8].

Based on data from *w3techs*, Nginx and Apache are the two most popular web servers, holding 34.1% and 30.5% of the market usage, respectively, as of December 31, 2023 [9]. Currently, more and more companies use Nginx as their web server [10]. This paper explores the reasons for Nginx's high market share from the perspective of resisting HTTP DDoS Flood attacks caused by large-scale concurrent Requests, which is also a major contribution of this work.

For this experiment, we used the Linux Ubuntu (22.04) and the latest Apache and Nginx versions to build a real test environment. For Apache, we utilized *mpm_worker* mode [11], known for its hybrid approach incorporating multi-Process and multi-Thread designs. This mode is tailored to optimize server performance while conserving system resources, offering an improvement over Process-based servers like Apache's *mpm_prefork* [12]. For this reason, *mpm_worker* was used as the mode to assess Apache's ability to resist HTTP DDoS Flood attacks.

Nginx is a Master Process and Multi-Worker Process design. Besides, Nginx uses an Event-Driven Asynchronous Non-Blocking architecture to efficiently handle a large number of connections at the same time [13].

In this experiment, we used a legitimate client to transmit normal HTTP Requests at 100 Threads per second to access our website on Apache and Nginx web servers. Threads are an integral part of the operating system. As an entity, it exists within a Process and is in charge of processing computer tasks, for example, sending and handling Requests in the HTTP communication [14].

We used 10,000 malicious Threads and 10,000 malicious HTTP Requests per thread traffic to attack the victim Apache and Nginx. Through multiple parameter attempts, we found that these are excellent numbers to investigate and compare the ability of two web servers, Apache and Nginx, to handle legitimate Requests during HTTP DDoS Flood attacks. In this paper, we collected and analysed changes in the following two metrics:

Transactions Per Second (TPS) represents Apache and Nginx web servers handling legitimate HTTP Requests per second in a given time [15] in our experiments. By measuring TPS, the change in the capacity of Apache and Nginx web servers to handle legitimate Requests can be assessed.

HTTP Requests Error Rate is calculated as the proportion of errored HTTP Requests to the total number of legitimate Requests recorded during testing [16]. *Apache JMeter* can configure connection and response timeouts for the HTTP Request. The connection timeout determines the duration at which an HTTP Request establishes a connection with a web server. The response timeout sets the interval for awaiting a response from the server to the client once a connection has successfully been made [17]. In our experiments, we set the above two timeouts to twelve seconds based on the tolerated waiting time for website users [18]. Therefore, when any timeout reaches the value we set, the legitimate HTTP Request will be considered a failed (errored) Request if the web server cannot process it due to insufficient resources.

The structure of the paper is organized as follows: The related literature is presented in Section two. Section three compares the architectural designs of Apache and Nginx. The experimental setup is detailed in Section four, while Section five covers the testing and results collection tools. A thorough analysis of the experimental findings is conducted in Section six. Section seven summarizes the results and findings of this paper and follows the future research works that be explored.

II. Related Work

In related work, we review studies in recent years by some researchers working on HTTP Flood attacks and evaluating and comparing web server performance.

In 2018, Guoliang Liu et al. [19] compared the performance of three popular web servers, Apache, Nginx and Lighttpd, in a 10G/40G network. They used the benchmark tool apib to test the server's throughput at different GET Request sizes of 1KB, 4KB, 16 KB, 64KB, and 256 KB. The results show that all evaluated servers can saturate the 40G network if the Request granularity is large enough. However, if the requested resource is less than 64KB, Nginx performs better than Lighttpd and Apache. Therefore, Nginx is the best web server for frequently accessing small files in big data applications.

In 2018, Zebari and colleagues [20] examined the effects of HTTP and TCP SYN Flood attacks on Apache 2 and IIS 10.0, using metrics such as average response time, CPU usage, and standard deviation to assess the servers' responsiveness, efficiency, and stability. The data revealed that under HTTP Flood attacks, IIS web server had greater efficiency and responsiveness. In contrast, fighting against TCP SYN Flood attacks, Apache displayed superior reactivity and stability.

In 2019, Hidayanto and Sawitri [21] performed a performance evaluation of e-commerce applications running on Apache and Nginx web servers. They used the web stress testing tool, *Apache JMeter*, to test the home page and registration page (10 scenarios) using different Threads and rate. They evaluated Apache and Nginx web server performance based on three collected metrics: response time, latency (delay time), and throughput.

In 2022, Farabi Fardin Khan et al. [22] proposed a resource-sharing-based method to mitigate DDoS attacks. In the experiment, the tools they used included *Wireshark*, *Apache JMeter*, and *Docker*. They use existing filtering methods to track attacker IP addresses, then send them to a proxy server and remove unwanted IP packet Requests through a memory management system. They used 50 Nginx virtual machine servers to simulate the blockchain network. Their experiment has proven their solution effective, as 50 virtual machines can mitigate 66% of attacks.

In 2021, Jader et al. [23] conducted a performance investigation of load balancing techniques in cluster-based web servers (round robin, least connections, and IPHash/Source). The test technique uses *Apache JMeter* and distributed technologies to simulate a high-volume load (100,000 to 500,000 HTTP Requests) in a real network. Experimental findings reveal that the suggested Nginx-based cluster is more responsive and reliable, consuming fewer resources in response time, throughput, standard deviation, and CPU Utilization metrics. In terms of error rate, the Apache-based cluster is efficient. Furthermore, the Round Robin method performs somewhat better for Nginx-based clusters. The IP-Hash method beat the other two algorithms in an Apache-based cluster in all measures.

In 2020, Parvinder Singh Saini and colleagues [24] used the machine learning tool WEKA to detect DDoS attacks. They used a dataset containing 27 features and five classes (Normal, UDP Flood, HTTP Flood, Smurf Attack and SID DoS). They used four machine learning algorithms: Naïve Bayes, Random Forest, Multi-layer Perceptron (MLP) and J48. Their research showed that J48 outperformed other algorithms with an accuracy of 98.64%.

In 2023, Tang et al. [25] evaluated a single Apache web server to resist HTTP Flood attacks by building a real test environment. The experiment shows that DDoS attacks cause the webserver to lose the ability to provide service for legitimate users faster than DoS attacks because more malicious Requests maliciously occupy the limited Thread resources of the web server. This experiment collected real web server performance metrics TPS and error rate of normal Requests. Our research team also have evaluated the impact of other DDoS attacks on network performance [5, 7, 26].

As we observe from above related works, no one has evaluated and compared the performance of the most recent Apache and Nginx versions during HTTP DDoS Flood attacks using a real testbed. This study aims to explore the reasons behind the rapid growth in Nginx market usage in recent years by generating new data and evaluating these two web servers' abilities to resist this attack.

III. Apache And Nginx Architecture Design

'Process' describes how a computer's CPU organizes resources in the operating system, while Threads can share resources allocated to Process. When a computer needs to handle multiple tasks simultaneously, a Process uses multiple Threads to achieve it [14]. In HTTP communication, the behaviour of users accessing a website, the web server receives HTTP Requests and will use its resources (e.g., Threads) to process these Requests. The HTTP DDoS Flood attack aims to exhaust the web server's Thread resources, preventing it from responding to normal Requests [8].

As we introduced earlier, Apache *mpm_worker* and Nginx adopt different design architectures. For the hybrid multi-Process and multi-Threaded Apache *mpm_worker*, one

Thread is responsible for one Request [11, 27]. However, as explained in Figure 1, Nginx uses an Event-Driven Asynchronous Non-Blocking Architecture with one master Process and multiple worker Processes. Each worker Process can handle multiple HTTP connections [13].

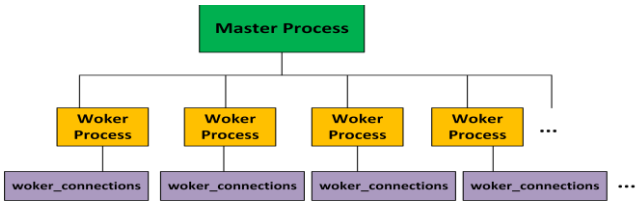

Fig. 1. Nginx Architecture [13].

Concurrency is the most important performance indicator for a web server since it represents the bottleneck in which the server can handle Requests at the same time [28]. In Apache *mpm_worker*, Concurrency is specified by the *'MaxRequestWorkers'* parameter [11, 27], which we set to be 2,000. This means that Apache can handle up to 2,000 Requests simultaneously. This concurrent value can support our website in the experiment.

However, Nginx adopts a different Concurrency design. In Nginx, *'worker_processes'* and *'worker_connections'* jointly specify the Nginx's Concurrency. *'worker_processes'* specifies the number of worker Processes Nginx has. *'worker_connections'* specifies the maximum number of connections that can be processed simultaneously [29]. For experimental comparison with Apache, we set *worker_processes* to 2 and *worker_connections* to 1,000. An HTTP connection can contain multiple HTTP Requests [30]. Therefore, Nginx can handle more Requests than Apache, even if both specify Concurrency values set to 2,000.

IV. Experimental Setup

Figure 2 shows the experimental testbed for building a real network. All devices support 1GB bandwidth and are connected via Category 6 cables to the legitimate client, four physical computers as attackers, the victim web server, the Cisco SG300 Layer 3 switch, and the Cisco 2911 router. The network is divided two subnets: the legitimate client and attackers are located at 192.168.1.0, and the victim web server at 192.168.2.0. Based on the above scenario, we replicated a real HTTP DDoS Flood attack case. This real and controllable experimental environment ensures the reliability of our results without violating cyber security ethics.

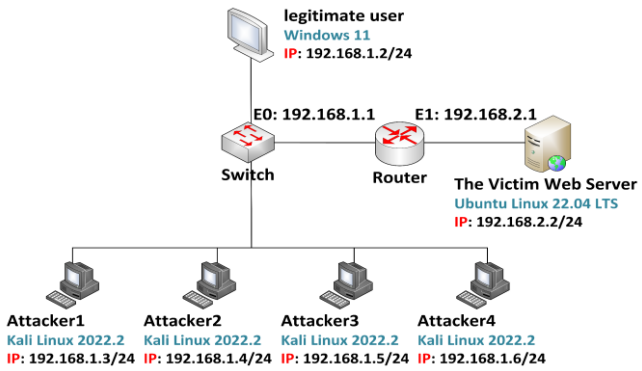

Fig. 2. Network Testbed.

On the victim server, we deployed and configured Apache (*mpm_worker*) and Nginx as web servers. The victim web server is equipped with an Intel i7 processor, 16GB of Random Access Memory (RAM), and is based on an Asus motherboard.

As for operating systems, the legitimate client has the latest Windows operating system (Windows 11) installed. *Apache JMeter* was used to collect the evaluated metrics on this legitimate client. The victim server was running Ubuntu (22.04), while all attack machines installed Kali Linux (22.2), an operating system designed for cybersecurity experts.

V. Legitimate And Malicious Traffic Tools

Apache JMeter [31] was installed on the experimental legitimate client (IP: 192.168.1.2). It sends legitimate HTTP Requests at 100 Threads per second to the target Apache (*mpm_worker*) and Nginx web servers and collected two key metrics: TPS and HTTP Requests Error Rate. This approach allowed us to replicate a legitimate scenario where 100 users visited our website per second.

Hibernet [32] is an HTTP DDoS Flood ethical hacking tool written in Python. We installed it on four attack computers (IP:192.168.1.3 to 192.168.1.6), and each attacker simultaneously sent 10,000 Threads and 10,000 Requests per thread attack to the victim web server from the sixth second. Six seconds is the minimum test granularity on *Apache JMeter*. For this reason, launching the attack at the sixth second can demonstrate changes in the two evaluated web server's ability to handle legitimate Requests without and during the attack.

In this experiment, each test lasted four minutes, and we repeated the testing and collection of each metric more than ten times up to the standard deviation divided by the average was less than 0.07. This avoids the randomness of each test result and ensures reliability. Furthermore, each test for four minutes is sufficient, as extending the test time does not affect the conclusions of our evaluation.

VI. Experimental Results

This experiment collected and analysed Transactions Per Second (TPS) in Section VI.A, and HTTP Requests Error Rate in Section VI.B.

A. Transactions Per Second (TPS)

In Figure 3, without attacks, Apache's average TPS was 10,765 during the overall test, while Nginx's TPS was 10,808. It means that Apache and Nginx can handle 10,765 and 10,808 legitimate HTTP Requests per second without attacks.

During attacks, for Apache, average TPS first shows a downward trend, from the 6th second to the 30th second, average TPS were measured as 10,734, 5,438, 406, 199 and 172. However, at the 36th second, TPS briefly increased to 578. The reason is that part of the Request processing is completed, and Apache's Threads resources are released, resulting in a temporary increase in the processing capacity of legitimate Requests (TPS). From the 48th second to the end, the average TPS averaged a low value of 8. This indicates that the victim Apache is overloaded and unable to respond to our legitimate client.

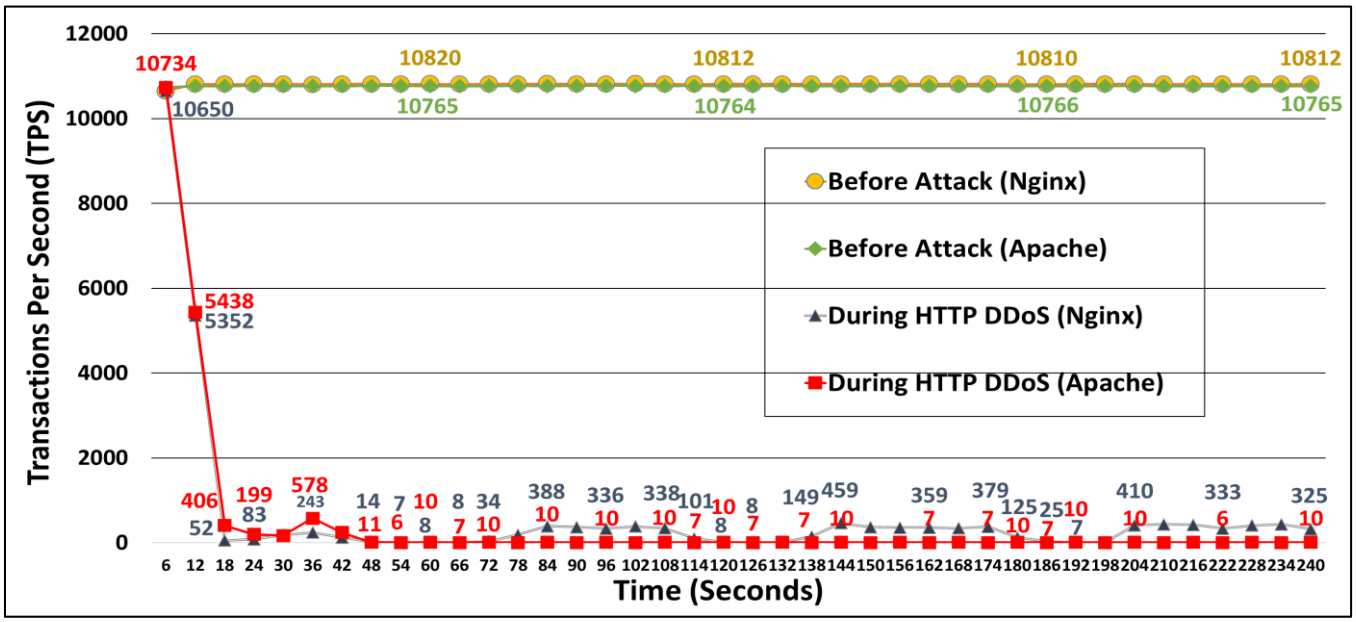

Fig. 3. Transactions Per Seconds (TPS) Before and During Attack on Apache and Nginx.

For Nginx, during the attack, the average TPS dropped from 10,650 at the 6th second to 5,352 at the 12th second and then to 52 at the 18th second. This decline indicates that Nginx's ability to handle legitimate Requests was significantly reduced due to attack. However, the average TPS of Nginx gradually picked up from the 24th second to the 36th second, with average TPS results of 83, 185, and 243 every 6 seconds, respectively. From the 42nd to the 66th second, the average TPS dropped again, with 123, 14, 7, 8, and 8 results. Then, at the 84th second, TPS rebounded slightly to 388. In our tests, Nginx's TPS always exhibited these ups and downs. Because 'Nginx's Better Concurrency Design' and 'Nginx's Asynchronous Non-Blocking Architecture' make Nginx more efficient than Apache when processing legitimate Requests during HTTP DDoS attacks. Therefore, the victim Nginx is not always overloaded. The reason for this is as follows:

- **Better Concurrency Design in Nginx:** The Nginx processing mechanism is connection-based, while Apache is Request-based (see Section III). An HTTP connection may include multiple HTTP Requests [30]. Therefore, Nginx may handle more Requests.
- **Nginx's Event-Driven Asynchronous Non-Blocking Architecture Design** [13]**:** Unlike traditional web servers, Nginx adopts an Event-Driven Asynchronous and Non-Blocking design. In short, this means that Nginx's worker Process can handle multiple HTTP connections at the same time. When one connection waits for a client response, Nginx's worker Process can quickly switch to other active HTTP connections without waiting idle. This is in sharp contrast to Apache's Blocking architecture, which refers to the fact that in most web servers (e.g., Apache's *mpm_worker* mode), each Thread can only process one Request and cannot process any other Requests until the current Request is completed [11, 13, 27]. Therefore, Nginx's ability is better than Apache's when processing Requests under attacks.

TABLE I. COMPARISON OF AVERAGE TRANSACTIONS PER SECOND (TPS) BETWEEN APACHE AND NGINX

Scenario	Time (Seconds)	Average TPS
Before Attack (Nginx)	**Four minutes**	10808
Before Attack (Apache)	**Four minutes**	10765
During Attack (Nginx)	**Four minutes**	606
	From the 48th second	229
During Attack (Apache)	**Four minutes**	451
	From the 48th second	8

Table I lists the average TPS results for the overall test period before and during the attack and from the 48th second to the end during the attack on Apache and Nginx. The reason we chose to observe the average TPS from the 48th second to the end is that Apache was overloaded from the 48th second until the end during the attack. In contrast, Nginx was not constantly overloaded (e.g., not overloaded from the 72nd second to the 114th second, from the 138th second to the 180th second, or from the 204th second to the end).

Without attacks, Apache's average TPS during the overall test was 10,765, while Nginx's TPS was 10,808. However, during attacks, throughout the testing period for Apache, the average TPS was 451. From the 48th second to the end, its average TPS was 8, indicating that the Apache web server could barely handle legitimate Requests due to insufficient Thread resources (*'MaxRequestWorkers'* [11, 27]).

In contrast, during attacks, throughout our test, Nginx's average TPS was 606. From the 48th second to the end, Nginx's average TPS was 229. It can be seen from these results that Nginx performs better than Apache in handling legitimate Requests when facing HTTP DDoS attacks.

B. HTTP Requests Error Rate

In Figure 4, without attack, neither Apache nor Nginx has any legitimate Requests with errors. All legitimate Requests are processed and responded to quickly by Apache and Nginx.

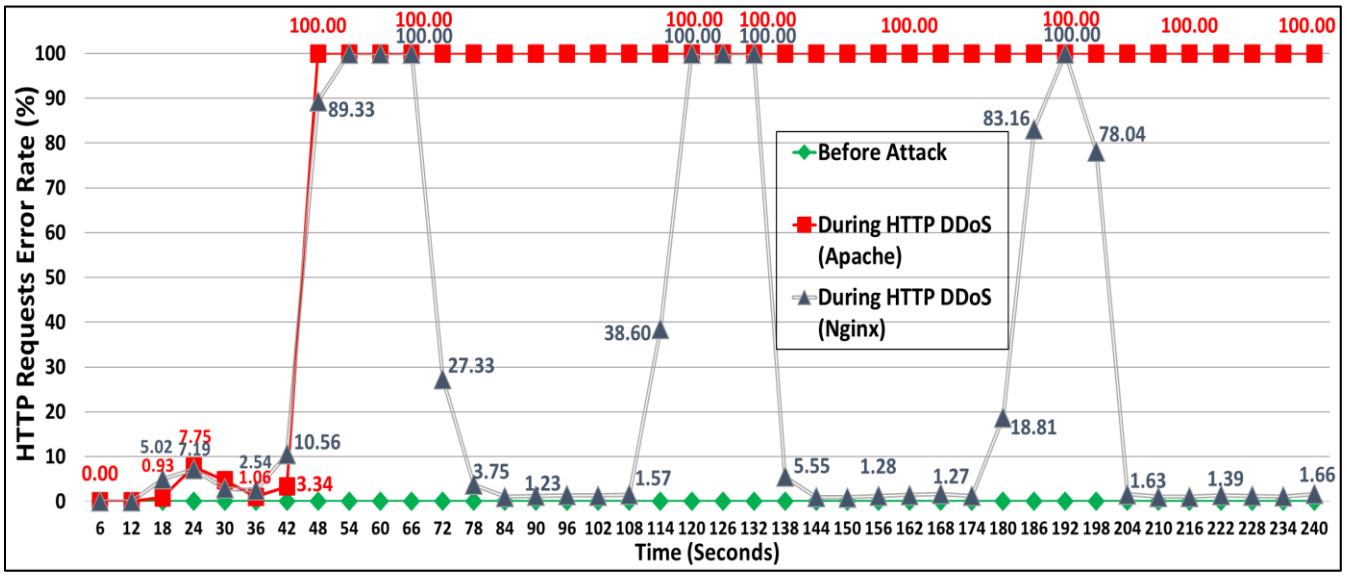

Fig. 4. HTTP Requests Error Rate Before and During Attack on Apache and Nginx.

However, under HTTP DDoS attacks, for both web servers, the HTTP Requests Error Rate increases. The HTTP Requests Error Rates of Nginx and Apache reached 7.19% and 7.75%, respectively, at the 24th second. The HTTP Requests Error Rate means that legitimate Requests cannot connect or be processed and responded by Apache or Nginx within the 12-second timeout [16, 17].

At the 36th second, under the attack, the HTTP Requests Error Rates of Nginx and Apache dropped to 2.54% and 1.06%, respectively. The reason for this temporary decrease is similar to the drop we discussed and reasoned in Section VI.A. The reason is that part of the Request processing is completed, and the web server's Threads resources are released, temporarily improving the speed of processing legitimate Requests, and reducing the number of errored legitimate Requests.

As the attack continues, from the 48th second to the end, Apache's HTTP Requests Error Rate remained at 100%. It indicates that Apache is always in an overloaded state. However, on Nginx, the HTTP Requests Error Rate for legitimate Requests exhibits fluctuations. For example, it reached 100% from the 54th to 66th second, dropped to 27.33% at the 72nd second, and dropped to 3.75% at the 78th second. At the 114th second, the HTTP Requests Error Rate increased to 38.60% and increased again to 100% at the 120th second. The reason why Apache and Nginx show differences in HTTP Requests Error Rates is that Nginx is not always as overloaded as Apache. As we analysed in Section VI.A, this is due to Nginx's inherent Concurrency design and Event-Driven Asynchronous Non-Blocking Architecture [13]. These two advantages make Nginx better able to handle concurrent Requests than Apache.

TABLE II. COMPARISON OF AVERAGE HTTP REQUESTS ERROR RATE BETWEEN APACHE AND NGINX

Scenario	Time (Seconds)	Average TPS
Before Attack (Apache and Nginx)	**Four minutes**	0%
During Attack (Nginx)	**Four minutes**	27.40%
	From the 48th second	32.36%
During Attack (Apache)	**Four minutes**	82.94%
	From the 48th second	100%

In Table II, before being attacked, the HTTP Requests Error Rate of Apache and Nginx were 0%; both could successfully process and respond to legitimate Requests. During HTTP DDoS attacks, Nginx's HTTP Requests Error Rate for the test period was an average of 27.40%, while Apache was an average of 82.94%.

However, from the 48th second that Apache gets overloaded to the end, Apache's HTTP Requests Error Rate was 100%, while Nginx's was an average of 32.36%. As reasoned above, Nginx is better than Apache in responding to legitimate Requests when attacked. This is consistent with our TPS observations in Section VI.A, proving that Nginx has better resilience for HTTP DDoS attacks than Apache.

VII. CONCLUSION

This study conducted an assessment and comparison of the effects of the most recent versions of Apache and Nginx web servers on Ubuntu Linux (22.04) when subjected to HTTP Flood attacks, producing new data in the process. The analysis examined Transactions Per Second and the HTTP Requests Error Rate, considering both the pre-attack and during-attack phases.

For parameters used, the results revealed that the HTTP DDoS Flood attack severely impacted the ability of Apache and Nginx to handle Requests from the normal user. In Section VI.A, without attacks, the average TPS of Apache during the entire test was measured as 10,765, while the TPS of Nginx was 10,808. However, during the attack, Apache measured an average TPS of 451 during the entire test period, while Nginx had an average TPS of 606. From the 48th second to the end, Apache's TPS for legitimate Requests remained at 8, while Nginx's TPS was 229. This shows that from the 48th second to the end, Apache had been overloaded and unable to respond to legitimate Requests.

In Section VI.B, when there were no attacks, Apache and Nginx successfully handled all legitimate Requests. However, during the attack, the average HTTP Requests Error Rate was 27.40% for Nginx and 82.94% for Apache. From the 48th second to the end, for Nginx, the HTTP Requests Error Rate for legitimate Requests was 32.36%, while for Apache, it remained 100%. This shows that Apache's all resources were occupied, and legitimate

Requests cannot be connected to Apache or processed and responded to within the 12 seconds we set. Nginx's lower HTTP Requests Error Rate shows it can still guarantee the successful processing and response of part legitimate HTTP Requests due its superior designs.

Our above experimental results show that Nginx can better resist HTTP DDoS Flood attacks than Apache due to reasons explained in this paper. Nginx's ability to better handle large numbers of concurrent Requests may have contributed to its increased market usage in recent years.

VIII. Future Work

We intend to investigate and assess the effectiveness of some measures to mitigate HTTP DDoS Flood attacks on various web servers, including Nginx, IIS, Apache and Lighttpd in the future. Also, future work may include other types of DDoS attacks and other operating systems. For example, DNS Flood or amplification attacks target Windows server operating systems.

References

[1] N. Boudriga, *Security of mobile communications*. Auerbach Publishers, Incorporated, 2009.

[2] 'Application layer DDoS attack'. Accessed: Mar. 26, 2024. [Online]. Available: https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/

[3] O. Yoachimik and J. Pacheco, 'DDoS threat report for 2023 Q3', The Cloudflare Blog. Accessed: Jan. 05, 2024. [Online]. Available: https://blog.cloudflare.com/ddos-threat-report-2023-q3

[4] R. Gupta, *Hands-on cybersecurity with blockchain: implement DDoS protection, PKI-based identity, 2FA, and DNS security using blockchain*. Packt, 2018.

[5] S. S. Kolahi, K. Treseangrat, and B. Sarrafpour, 'Analysis of UDP DDoS flood cyber attack and defense mechanisms on Web Server with Linux Ubuntu 13', in *2015 International Conference on Communications, Signal Processing, and their Applications (ICCSPA'15)*, Sharjah, United Arab Emirates: IEEE, Feb. 2015, pp. 1–5. doi: 10.1109/ICCSPA.2015.7081286.

[6] M. Prince, 'Deep Inside a DNS Amplification DDoS Attack', The Cloudflare Blog. Accessed: Dec. 31, 2023. [Online]. Available: https://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack

[7] S. S. Kolahi, A. A. Alghalbi, A. F. Alotaibi, S. S. Ahmed, and D. Lad, 'Performance comparison of defense mechanisms against TCP SYN flood DDoS attack', in *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, St. Petersburg, Russia: IEEE, Oct. 2014, pp. 143–147. doi: 10.1109/ICUMT.2014.7002093.

[8] K. Singh, P. Singh, and K. Kumar, 'Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges', *Comput. Secur.*, vol. 65, pp. 344–372, Mar. 2017, doi: 10.1016/j.cose.2016.10.005.

[9] 'Usage Statistics and Market Share of Web Servers, December 2023'. Accessed: Dec. 31, 2023. [Online]. Available: https://w3techs.com/technologies/overview/web_server

[10] M. Gelbmann, 'Nginx is now the most popular web server, overtaking Apache'. Accessed: Dec. 31, 2023. [Online]. Available: https://w3techs.com/blog/entry/nginx_is_now_the_most_popular_web_server_overtaking_apache

[11] 'worker - Apache HTTP Server Version 2.4'. Accessed: Dec. 31, 2023. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/worker.html

[12] 'prefork - Apache HTTP Server Version 2.4'. Accessed: Dec. 31, 2023. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/prefork.html

[13] O. Garrett, 'Inside NGINX: Designed for Performance & Scale', NGINX. Accessed: Jan. 04, 2024. [Online]. Available: https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/

[14] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts*, Ninth edition. Wiley, 2012.

[15] 'Apache JMeter - User's Manual: Glossary'. Accessed: Dec. 31, 2023. [Online]. Available: https://jmeter.apache.org/usermanual/glossary.html#Throughput

[16] 'Aggregate_Report'. Accessed: Dec. 31, 2023. [Online]. Available: https://jmeter.apache.org/usermanual/component_reference.html#Aggregate_Report

[17] 'HTTP Request'. Accessed: Dec. 31, 2023. [Online]. Available: https://jmeter.apache.org/usermanual/component_reference.html#HTTP_Request

[18] J. A. Hoxmeier and C. DiCesare, 'System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications', in *AMCIS 2000 Proceedings*, 2000. [Online]. Available: https://aisel.aisnet.org/amcis2000/347

[19] G. Liu, J. Xu, C. Wang, and J. Zhang, 'A Performance Comparison of HTTP Servers in a 10G/40G Network', in *Proceedings of the 2018 International Conference on Big Data and Computing*, Shenzhen China: ACM, Apr. 2018, pp. 115–118. doi: 10.1145/3220199.3220216.

[20] R. R. Zebari, S. R. M. Zeebaree, and K. Jacksi, 'Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers', in *2018 International Conference on Advanced Science and Engineering (ICOASE)*, Duhok: IEEE, Oct. 2018, pp. 156–161. doi: 10.1109/ICOASE.2018.8548783.

[21] R. Hidayanto and P. Sawitri, 'Performance Testing of e-Payment Website Using JMeter', *International Research Journal of Advanced Engineering and Science*, vol. 4, no. 3, pp. 350–352, 2019.

[22] F. F. Khan, N. M. Hossain, Md. N. H. Shanto, S. B. Anwar, and J. Noor, 'Mitigating DDoS Attacks Using a Resource Sharing Network', in *Proceedings of the 9th International Conference on Networking, Systems and Security*, Cox's Bazar Bangladesh: ACM, Dec. 2022, pp. 1–11. doi: 10.1145/3569551.3569560.

[23] O. H. Jader *et al.*, 'Ultra-Dense Request Impact on Cluster-Based Web Server Performance', in *2021 4th International Iraqi Conference on Engineering Technology and Their Applications (IICETA)*, Najaf, Iraq: IEEE, Sep. 2021, pp. 252–257. doi: 10.1109/IICETA51758.2021.9717748.

[24] P. S. Saini, S. Behal, and S. Bhatia, 'Detection of DDoS Attacks using Machine Learning Algorithms', in *2020 7th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India: IEEE, Mar. 2020, pp. 16–21. doi: 10.23919/INDIACom49435.2020.9083716.

[25] H. Tang, S. S. Kolahi, and L. Thompson, 'Evaluation of HTTP Flood DDoS Cyber Attack on Apache2 Web Server with Linux Ubuntu 22.04', in *2023 IEEE International Conference on Computing (ICOCO)*, Langkawi, Malaysia: IEEE, Oct. 2023, pp. 53–58. doi: 10.1109/ICOCO59262.2023.10398152.

[26] S. S. Kolahi, B. Barmada, and K. Mudaliar, 'Defence mechanisms evaluation against RA flood attacks for Linux-victim node', in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Kuala Lumpur: IEEE, Dec. 2017, pp. 1000–1005. doi: 10.1109/APSIPA.2017.8282169.

[27] 'MaxRequestWorkers Directive'. Accessed: Dec. 31, 2023. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/mpm_common.html#maxrequestworkers

[28] T. Ma, C. Wu, W. Tian, and W. Shen, 'The performance improvements of highly-concurrent grid-based server', *Simulation Modelling Practice and Theory*, vol. 42, pp. 129–146, Mar. 2014, doi: 10.1016/j.simpat.2013.12.008.

[29] R. Nelson, 'Tuning NGINX for Performance', NGINX. Accessed: Dec. 31, 2023. [Online]. Available: https://www.nginx.com/blog/tuning-nginx/

[30] S. Arya, 'HTTP Non-Persistent & Persistent Connection'. Accessed: Dec. 31, 2023. [Online]. Available: https://www.scaler.com/topics/persistent-connection-http/

[31] 'Apache JMeter - Apache JMeter™'. Accessed: Dec. 31, 2023. [Online]. Available: https://jmeter.apache.org/

[32] 'Hibernet'. Accessed: Dec. 31, 2023. [Online]. Available: https://github.com/All3xJ/Hibernet