FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 34, No 4, December 2021, pp. 525 - 546
https://doi.org/10.2298/FUEE2104525G

Original scientific paper

AUTOSCALABILE DISTRIBUTED ANTI-SPAM SMTP SYSTEM BASED ON KUBERNETES

Nadja Gavrilović, Vladimir Ćirić

University of Niš, Faculty of Electronic Engieering, Niš, Serbia

Abstract. *Due to the increasing amount of spam email traffic, email users are in increasing danger, while email server resources are becoming overloaded. Therefore, it is necessary to protect email users, but also to prevent SMTP system overload during spam attacks. The aim of this paper is to design and implement an autoscalable distributed anti-spam SMTP system based on a Proof of work concept. The proposed solution extends SMTP protocol in order to enable the evaluation of the client's credibility using the Proof of work algorithm. In order to prevent resource overload during spam attacks, the anti-spam SMTP system is implemented in a distributed environment, as a group of multiple anti-spam SMTP server instances. Kubernetes architecture is used for system distribution, configured with the possibility of autoscaling the number of anti-spam SMTP server instances depending on the system load. The implemented system is evaluated during a distributed spam attempt, simulated by a custom-made traffic generator tool. Various performance tests are given: (1) The proposed system's impact on client's behaviour and the overall amount of spam messages, (2) The performance of the undistributed anti-spam SMTP server during spam attack, in terms of resource load analysis (3) Autoscaling demonstration and evaluation of proposed distributed system's performance during a spam attack. It is shown that the proposed solution has the possibility of reducing the amount of spam traffic, while processing tens of thousands of simultaneous SMTP client requests in a distributed environment.*

Key words: *anti-spam, spam, email, smtp, kubernetes, proof of work*

Received March 25, 2021; received in revised form September 22, 2021
Corresponding author: Nadja Gavrilović
Faculty of Electronic Engineering, Computer Science Department, Aleksandra Medvedeva 14, 18000 Niš, Serbia
E-mail: nadja.gavrilovic@elfak.ni.ac.rs

1 Introduction

According to recent reports [1], approximately 50% of the overall email traffic consists of spam. Spam represents the abuse of electronic systems and SMTP protocol for the purpose of sending mass unwanted messages. Spam can lead to serious attempts at a data breach or email users identity theft, also. Furthermore, spam traffic is wasting valuable network and device resources [2].

There are a lot of different anti-spam solutions and techniques recently developed and applied in many different environments. In recent years, many of them relate to the problem of email spam [2, 3], spam in social networking [4–6], etc. One of the main approaches in filtering spam messages is based on the analysis of their contents. In the beginning, these solutions were based on a set of user-defined rules. In recent years, there were many proposed classification techniques, based on machine and deep learning [7,8]. These solutions can be time-consuming, but also can encounter a false alarm problem. Furthermore, spam attackers find ways to manipulate contents and structure of emails, so they can avoid content-based spam filters.

Another type of anti-spam solutions are reputation-based systems, which analyse the identity of message sender. There are many different approaches. For example, [9] verifies client's identity by using public key cryptography. Instead, this paper will propose solution which enables the evaluation of individual client's credibility using the Proof of Work (PoW) algorithm. Numerous papers were published which studied the implementation of this algorithm in various systems [10–12]. In general, the PoW algorithm can be applied to any automated malicious online interaction problem (e.g. spam comments, spam text messages), but it is important to take into account the specific protocol, implementation and backward compatibility. Additional communication is not always possible, or can be difficult to implement. To the best of our knowledge, only in [10] the authors suggested the use of the PoW concept as an anti-spam solution. They implemented an anti-spam system on a Peer to Peer (P2P) network. The system from [10] differs from the system proposed in this paper in terms of how the PoW algorithm is implemented. Also, the anti-spam solution proposed in this paper is based on client/server communication in a distributed environment, instead of the P2P system.

The goal of this paper is to design and implement a distributed anti-spam SMTP system based on a Proof of work concept. The proposed system does not address the problem of receiving spam messages on the client, but affects the first and biggest step in the existence of spam traffic - sending a large

number of spam messages in a short period of time. In order to enable PoW evaluation of the SMTP client, the extension of the SMTP protocol is designed. A common problem during spam attacks is server resource overload, which can cause denial of service. In order to prevent resource overload of the proposed solution, the anti-spam system is implemented in a distributed environment. Kubernetes architecture is used for distribution of multiple anti-spam SMTP server instances, which make up the proposed anti-spam SMTP system. Kubernetes is configured with the possibility of autoscaling the number of SMTP server replicas depending on the system load. The implemented system is evaluated during the distributed spam attempt, simulated by the custom made traffic generator tool. Various performance tests are given: (1) The client's perspective on the proposed system and the impact on the overall amount of spam messages, (2) The undistributed anti-spam server's performance during the spam attempt in terms of resource load analysis, (3) Autoscaling demonstration and distributed system's performance evaluation during spam attack. It is shown that the proposed solution significantly reduces the amount of spam traffic, while processing tens of thousands of simultaneous SMTP client requests in a distributed environment.

The paper is organized as follows. Section 2 gives a brief introduction to SMTP and Proof of Work algorithm. Section 3 is devoted to process of containerization and Kubernetes orchestration. Section 4 is the main section and presents the design and architecture of the proposed anti-spam SMTP system. Section 5 is devoted to the system evaluation during simulated spam attempt, while the concluding remarks are given in Section 6.

2 Background on SMTP and Proof of Work

The SMTP protocol defines rules for sending and reliable transfer of email messages through the network. The devices taking part in the transfer itself are referred to as agents and their communication is defined by the SMTP protocol.

Within communication of each two agents during email transmission, the device sending the email has the role of SMTP client, while the device which receives the email has the role of SMTP server. An email transaction begins with the command MAIL, which the client uses to define the email address of the sender. The second step is the definition of the email address of the recipient using the command RCPT, etc. The responses from the server ensure the synchronization of the activities in the SMTP client-server communication and provide the information on the success of client's actions.

The basic communication between the SMTP client (C) and server (S) during the successful transfer of a message is as follows [13]:

S: 220 smtpServer.example.com Simple Mail Transfer Service Ready
C: HELO smtpClient.example.com
S: 250 smtpClient.example.com, pleased to meet you
C: MAIL FROM: <from@example.com>
S: 250 Sender OK
C: RCPT TO:<recipient1@example.com>
S: 250 Recipient OK
C: DATA
S: 354 Enter mail, end with '.' on a line by itself
C: From: "From Example" <from@example.com>
C: To: Recipient Example <recipient@example.com>
C: Date: Tue, 15 May 2020 16:02:43
C: Subject: Hello
C: Hello world!
C: .
S: 250 Queued mail for delivery
C: QUIT
S: 221 Service closing transmission channel

Proof of Work systems emerged with the aim of verifying device's credibility and preventing abuse of computer's processing power, for example during Denial of Service (DoS) attacks. The basic idea behind this concept is to request relatively small amount of processing time from a device which tries to access protected resource, with the aim of fulfilling the criterion issued by the server. This prevents the underlying characteristic of aforementioned attacks - a large number of access attempts at a resource over a short period of time, without significantly affecting normal use of the system [14].

The principle behind how a PoW system works is based on a typical cryptographic scenario, during which the client who is requesting a service or resource attempts to prove its credibility to the server [14]. Most often, execution of mathematical or cryptographic functions represents a way of investing sufficient processing time as proof of credibility. The functions are not overly demanding, but are complex enough to ensure, in the case where their multiple execution is required, significant processing time on the client's side. The task of the client is to solve time-intensive calculation involving certain data many times over, until the obtained solution satisfies the requirement issued by the server. After the client sends that solution, the task of the server is to check its validity and identify the client as reliable or

unreliable. A very important criterion of PoW implementation is that it has to be difficult and time-consuming to calculate the solution on the client's side, while the solution validation on the server side does not require a large amount of time [11].

3 Containerization and Kubernetes Background

When developing software solutions, one of the common requirements is portability between different platforms and environments. Also, the ability to distribute and scale the developed application is an important prerequisite for its efficient and reliable execution. Recently, these requirements are commonly being met by introducing containerization into the software development process [15].

3.1 Docker Containerization

Today's application containerization technologies are based on research and solutions designed years ago, in the field of virtualization. Virtualization enables isolation of applications, but at the cost of significant impact on host resources. In order to overcome these disadvantages, containers are presented. They provide functionality similar to those provided by the virtual machines, while preserving host resources [16]. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

All containers running on a single device share its operating system, which frees up significant amounts of device resources, in comparison to virtual machines. Also, containers are easily portable between different devices and cloud platforms [15, 17].

The most commonly used container technology is Docker. Docker represents a set of software tools used for the development and distribution of software packages, i.e. containers. Docker technology is used for packaging the application and its runtime environment into a container, which can then be executed on many platforms [18].

3.2 Kubernetes Orchestration

When developing software solutions that require a reliable and scalable system that will balance the application load, the use of distributed container systems is becoming more common. In such situations, it is necessary to

ensure automatic container communication and management using orchestration tools. Since its introduction in 2014, Kubernetes orchestrator has grown to be one of the largest and most popular open source projects in the world. It has become the standard API for building distributed cloud-native applications, especially in big enterprises. Kubernetes enables the distributed execution of software applications on a large number of separate nodes that make up the Kubernetes cluster. The Kubernetes orchestration enables the automation of all aspects of container coordination and management and involves the process of placing, managing, scaling and connecting containers. The Kubernetes cluster consists of master and worker nodes. The master node (also called the Control Plane) manages the entire Kubernetes system, while the worker nodes are responsible for the execution of containerized applications. Worker nodes are Linux hosts which constantly listen for new tasks and execute assigned ones, and report state and changes to the master node [19, 20]. In order to define how the containerized applications are executed on the Kubernetes cluster, Kubernetes objects are created at the request of the user. The main Kubernetes object and the basic building block of the Kubernetes system is a Pod. A Pod can be defined as a shared environment for the execution of one or more containers. It can be seen as a wrapper around the container, which is necessary for the container execution on the Kubernetes cluster. Each Pod is executed exclusively on one node [20]. The Deployment Kubernetes object provides the ability to easily create and manage multiple Pods. It provides additional Pod features, which they do not originally have - self-healing, scalability, automatic rolling update and rollback mechanism. In order to have a constant access point to the containerized application that executes as a group of Pods, the Kubernetes Service object could be used. The Service is associated with the Pods and fronts them with a stable IP, DNS, and port. It also load-balances requests across the Pods.

The client does not have to know the exact location and configuration of the Pods, which can be unstable, change their location and network settings. By using Service object, the Pods can be scaled, new Pods can be started and previous versions of the Pods can be updated and shut down. The Service object in front of the Pods observes the changes, and maintains a list of active Pods ready to accept client connections. If external access to Service is needed, the NodePort Service type should be used, because it exposes the Service on each Node's IP address and a static port, and makes it possible to contact the Service from outside the cluster.

4 Design of autoscalable distributed anti-spam SMTP system

The proposed anti-spam SMTP server, which uses PoW algorithm for SMTP client verification, is implemented as a .NET Core application. The implemented application was containerized using Docker technology in order to enable horizontal scaling of lightweight, portable containers, which can run virtually anywhere. The containerized anti-spam SMTP server is distributed on the Kubernetes cluster in the cloud. Also, Kubernetes is configured with the possibility of autoscaling the number of required server replicas depending on the system load.

The role of the proposed distributed anti-spam SMTP system is shown in Fig. 1. The basic behaviour of the proposed solution is defined by the SMTP protocol. Its basic network functionalities, as a device that forwards an email to its destination (Fig. 1), have been upgraded with the implementation of a distributed anti-spam system that has the possibility of verifying SMTP clients, by asking for proof of their credibility. The proposed anti-spam system has the role of a proxy, which checks SMTP clients before forwarding their messages through the network, and marks the messages as spam if needed (Fig. 1). It uses PoW algorithm to check the credibility of email clients, by issuing a challenge that an SMTP client has to resolve by using its processing power. Marking is done by adding the field *X-spam-category: spam* to the header of the email message. In this way, the proposed anti-spam SMTP system alerts the next SMTP server to which it forwards the message about potential spam traffic, while containing all of the client's processing power which could be used for spamming. The role of the proposed solution is to prevent further passage of spam emails through the network, but also to slow down spam attacks, as will be described below.

A potential problem of the proposed anti-spam system could occur when it is simultaneously abused by multiple clients, which can deplete its resources because of a large number of parallel connections (a problem close to DoS attacks). Taking that into consideration, the proposed anti-spam system is designed to consist of many anti-spam SMTP server replicas (Fig. 1, replica 1 - replica N) distributed on the cluster, which can balance the system load if necessary. Also, the proposed system is configured with the possibility of autoscaling the number of SMTP server replicas depending on the system load.

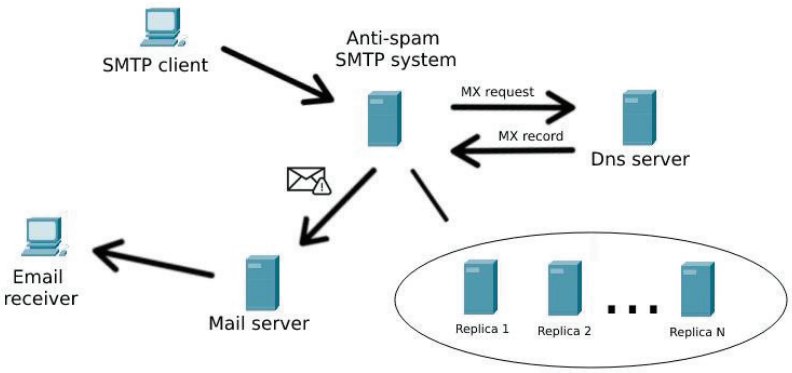

Fig. 1: The role of the proposed anti-spam system

4.1 The proposed Kubernetes architecture

The proposed solution has been distributed on Kubernetes, using Docker as containerization technology. The goal of distributed execution of the proposed anti-spam system is a reliable, fault-tolerant SMTP system, which is resistant to spam attacks and has higher total available resources.

The proposed Kubernetes architecture in which anti-spam system executes is shown in Fig. 2. In order to simplify the architecture shown, only two worker nodes are given. The figure shows a NodePort service named smtp-service, created to allow external clients to access the proposed anti-spam system. As can be seen, the proposed system is exposed on any of the worker node IP addresses, in combination with NodePort defined value of 30001. The range of values that NodePort on the Kubernetes architecture can have is from 30,000 to 32,767. In an SMTP production environment, the destination port of the incoming traffic could be easily changed on the router/firewall, from a standard TCP port value of 25 to the NodePort value of 30001. The client's request is forwarded from the node to the created smtp-service object. Then, after analyzing the cluster and available running Pods, the smtp-service can reroute the request to one of the smtp-pod objects, taking into account the current load of active Pods. Requests are forwarded on configured internal port 5000.

The number of currently active Pods depends on autoscaling component, which estimates current system load and creates additional Pods or shuts some of them down if needed. Autoscaling enables automatic regulation of the number of anti-spam SMTP server instances, depending on the system's

load, ie. the number and frequency of SMTP client requests. The configuration of the proposed Kubernetes system is done so that initially there is one anti-spam SMTP server instance, which is not replicated until the system's load indicates the need for a larger number of SMTP Pods that will serve all of the client requests. The moment the system load reaches a configured threshold, the system scales by adding new replicas.

The proposed system is autoscaled depending on two criteria - CPU and memory usage. In particular, autoscaling will be performed when the average Pods memory usage is greater than configured. Also, the number of replicas will be increased when the average Pod CPU utilization reaches 50%. When multiple metrics are specified, Kubernetes monitors both parameters and performs the addition of new replicas when either of them reaches the maximum defined value.

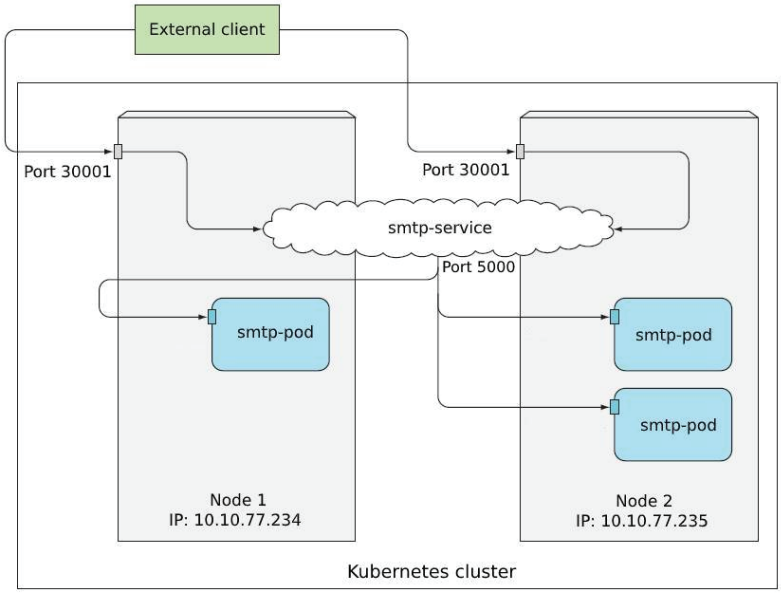

Fig. 2: The proposed Kubernetes architecture

4.2 The role of the proposed system

The proposed solution filters traffic and marks messages as spam if it is necessary. The SMTP client's credibility check is performed immediately after the proposed system receives the email. It has to decide whether the

client is a potential spammer, and if so, how it will be challenged using the PoW algorithm. The operation of the proposed system when processing a client's email request is presented by the algorithm in Fig. 3. The initial client evaluation can be performed using a reputation system. If the client is estimated as reliable, SMTP communication continues without any changes. An email message is sent through the network, to its destination. An alternative flow of the communication occurs in the event that during the

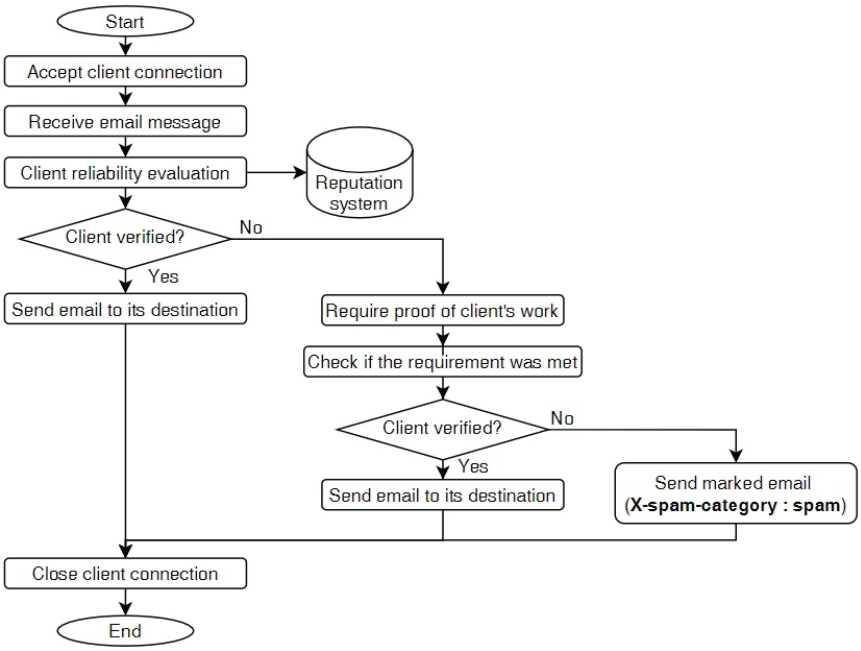

Fig. 3: The proposed anti-spam system algorithm, after receiving email request

communication, the client is estimated as a potential spammer. In that case, its validation will be performed by issuing a challenge that the client has to solve. If the client proves its validity, his email message will be forwarded further through the network, without any marking of the message. However, if the client does not prove its credibility, he is evaluated as a spammer, his email message is marked as spam traffic and his spam attempt is slowed down. The precise flow of communication between the SMTP client and the proposed anti-spam system, during the PoW algorithm, will be given in the next subsection.

4.3 PoW algorithm implementation

The use of PoW algorithm gives the possibility of asking a potential spammer for proof of credibility by requiring a certain amount of processing time. The implementation of such a system requires certain changes in the implementation of the SMTP protocol.

At the beginning of the communication, the SMTP client provides the server with information on the sender, recipient and overall message being sent, as defined in the basic SMTP protocol. The server, after it has received all the email data, has the possibility of transferring the email message to its destination, the same way it does in protocol-defined communication in the case of a valid consumer. However, if based on the evaluation of the sender, the reputation system determines that the client might be unreliable, in this paper we propose for SMTP protocol to require proof of credibility prior to the transfer of the messages. With that aim in mind, changes were made to the basic SMTP communication. The extended SMTP protocol is shown in Fig. 4.

```
 1: C: DATA
 2: S: 354 Enter mail, end with  "." on a line by itself
 3: C: From: From Example <from@example.com>
 4: C: To: Recipient Example <recipient@example.com>
 5: C: Date: Tue, 03 March 2020 16:02:43
 6: C: Hello world!
 7: C: .
 8: S: 250 Challenge number 03 md5_sha1_sha256_sha512
 9: C: Nonce: 79745 md5
10: S: 250 Queued mail for delivery
11: C: QUIT
12: S: 221 Server closing connection
```

Fig. 4: Extended SMTP protocol

The changes enable the server to send integer value, which represents the weight (line 8 in Fig. 4). Its value determines the criterion which the client has to meet. Along with the weight, the server sends the supported hash algorithms (line 8 in Fig. 4). The client matches one of the acceptable hash functions and computes it against the entire mail message, including the header and the time stamps. By computing a hash function on the entire email message, the client is required to generate a sequence that has as many zeroes in the beginning, as was defined by the previously received

weight parameter (line 8 in Fig. 4). The execution of the same hash function on the same data sequence always results in the same output. That is why the client has to append a nonce value to the data, based on which the client calculates the hash value of the message. The only way to determine a nonce value that satisfies the requirement of the server is brute force. The hash function is sequentially executed several times, until the generated output meets the server requirement, and with each function execution, the value of the nonce changes.

A single execution of a hash algorithm on the data does not require significant processing time. However, obtaining a satisfactory output sequence is sufficiently rare to take away significant CPU time from the client. Once the nonce value used to satisfy the issued requirement is found, the client forwards it to the server, along with the hash function which has been used (line 9 in Fig. 4). After receiving the nonce value from the client, it checks it to ensure if the obtained value meets the set requirement. The check is achieved through the execution of a single hash function on a previously obtained email in combination with a recently received nonce value. It can be noticed that a significantly greater amount of processing time is required to solve the given problem on the client than the amount of processing time required to verify the solution on the server. If the server determines that the applied nonce value meets the previously set requirement, the email is successfully transferred.

A valid client, unlike a spammer, rarely sends great number of email requests over a short period of time. Thus, even if the valid email client has been incorrectly evaluated by a reputation system as potentially dangerous, the processing time needed for sending a small number of emails will not render the use of the email service more difficult. Thus, valid user's use of the mail server has not undergone any noticeable changes. On the other hand, a spammer who is trying to send great amount of emails, will have to prove his credibility by executing PoW algorithm for each email message, which will result in a significant CPU time. It is important to note that the hash function is applied over the entire email (body and header), including the recipient and the time stamp. In this way, due to the time stamp in the header, it is assured that hash needs to be computed by the spammer every time, even if the same message is sent over and over again.

The example given in Fig. 4 shows the successful PoW validation of the email client. The server requires CPU time from the client by sending the Challenge number with the value of 3, which denotes that the client must generate an output sequence which in the beginning has precisely 3 zeroes (line 8 in Fig. 4). In the given example, the value of the nonce parameter is

79745 (line 9 in Fig. 4). It denotes that the value 79745, appended to the data which make up the content of the email message, is the piece of data that met the given requirement of the server.

Let us note that the proposed SMTP extension is backward compatible with the original SMTP specification. In the case of a client who does not support PoW, he will interpret the message containing the weight as a confirmation of successful sending of email due to code 250, and close the connection. In the currently implemented system, such an email will not be sent. One of the ideas for future work is to implement marking of this type of clients, so that they would be allowed to send a certain number of emails without using the PoW algorithm, as long as their activity does not indicate the possible presence of spam.

5 Implementation results

The proposed system is implemented and evaluated on the cloud, which consists of six servers, with a total RAM capacity of cca. 400GB. Five servers are IBM System x3550 models, with Intel (R) Xeon (R) CPU E5603 @ 1.60GHz processor type, with 8 logical processors. One server is HP ProLiant DL380p Gen8 model, with Intel (R) Xeon (R) CPU E5-2620 v2 @ 2.10GHz processor type, with 12 logical processors. The Kubernetes system is implemented on Linux Ubuntu 16.04 virtual machines. Each machine has 2 logical cores and 8GB of RAM. The Kubernetes cluster consists of five virtual machines of equal resources, one of which is the master node, while the other four are worker nodes.

The experiment is set as follows: 1) an evaluation of the client's effort and the impact on the overall amount of sent spam messages, 2) an evaluation of the spam attack impact on undistributed anti-spam SMTP solution, 3) the system performance during the spam attempt when the SMTP solution is distributed on the Kubernetes architecture. Also, automatic autoscaling of the proposed solution, depending on the system load, will be demonstrated. The evaluated client/server communication and the hash function used are as given in Fig. 4. In order to simulate distributed spam attack and maximize the system load, a custom traffic generator tool with the PoW support was made.

5.1 The client's perspective

Evaluation of client's work for different weight values, is discussed in [21]. The results obtained in [21] are given in Table 1.

Table 1: An evaluation of the client work for various weights

Weight	Avg. time	Standard deviation	Avg. # of hash functions	# of completed tests
1	1.14ms	0.41ms	41	20
2	28.7ms	25.09ms	1728	20
3	4.35s	2.88s	223620	20
4	2.71min	1.81min	10996000	20

Furthermore, this paper gives the evaluation of client's behaviour and the PoW impact on the amount of spam traffic.

It is determined that with PoW algorithm, the number of emails sent from the client per second does not depend on the number of sent requests from the client per second. It depends only on the value of the weight given by the server and the client's processing power, as in:

$$N = \frac{HP}{H(T)}, \tag{1}$$

where N is the number of outbound messages from the client per second, HP is the hash power of the client, T is the weight parameter, and H(T) is the average number of required executions of the hash algorithm for the weight T. The reason is that with the increase in the number of simultaneously initiated client SMTP requests, the number of executed hash functions per connection per second decreases due to the increase in CPU load. A direct consequence of reducing the number of hash function executions per connection is an increase in the duration of an individual client/server connection. We can conclude that a client with the intention of abusing the email server is limited to a constant number of sent emails per second, which is slowing down his spam attacks [21].

In addition, in this paper we will design and implement an anti-spam SMTP system based on a discussed Proof of work concept, which is distributed by using Kubernetes architecture, with the possibility of autoscaling the number of SMTP server instances depending on the system load.

5.2 The performance evaluation of the undistributed anti-spam SMTP server

In order to evaluate the system with the increased load, aforementioned distributed spam attack is simulated using 10 virtual machines. A custom

generator of a large amount of SMTP requests for sending email is implemented, which supports the PoW algorithm. In order to load the system with as many parallel connections as possible, the spam generator tries to send 10,000 emails in a row. Fig. 5 shows the client processor load before and during the attack. It can be noticed that from the moment when the client starts sending SMTP requests, its cpu usage is 100%.

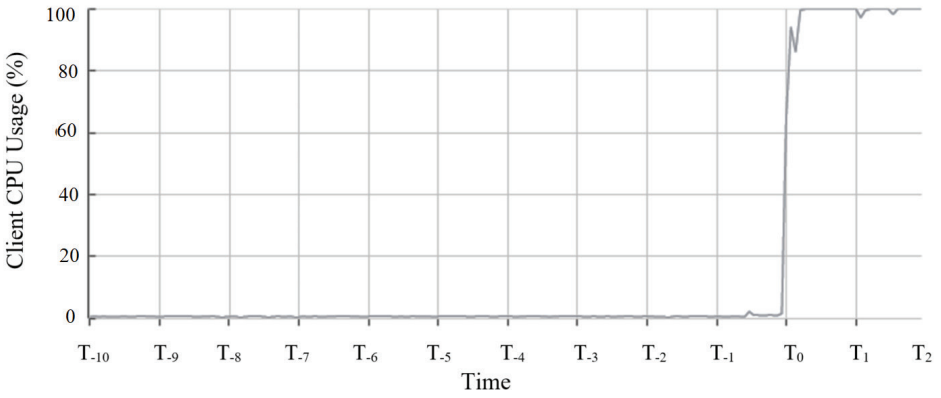

Fig. 5: Individual client CPU load before and during the spam attack

Fig. 6 shows the amount of network traffic generated by the individual client who performs the spam attack. At the beginning of the attack, the client initiates large number of connections at the same time, creating a thread for each email. It can be seen that at the time of the attack beginning (T0), the client generates traffic of approximately 5MBps. Within each connection, the proposed PoW system will request CPU time from the client, which will result in a client CPU load (Fig. 5). The client, relatively quickly after the start of the attack, opens the maximum number of connections that its resources can support (because each connection implies one thread that significantly loads the CPU by sequentially executing hash functions). When client's resources become significantly loaded, he is no longer able to send the initial amount of data, which can be seen in the graph, after the moment T0. Nevertheless, the client continues to send a relatively small amount of data to the server, as he solves the challenges one by one (Fig. 6, between moments T0 and T1).

Fig. 7 shows the dependence of the proposed anti-spam SMTP server CPU usage on the number of clients simultaneously performing the spam attack. It can be noticed that the CPU of the server is not significantly

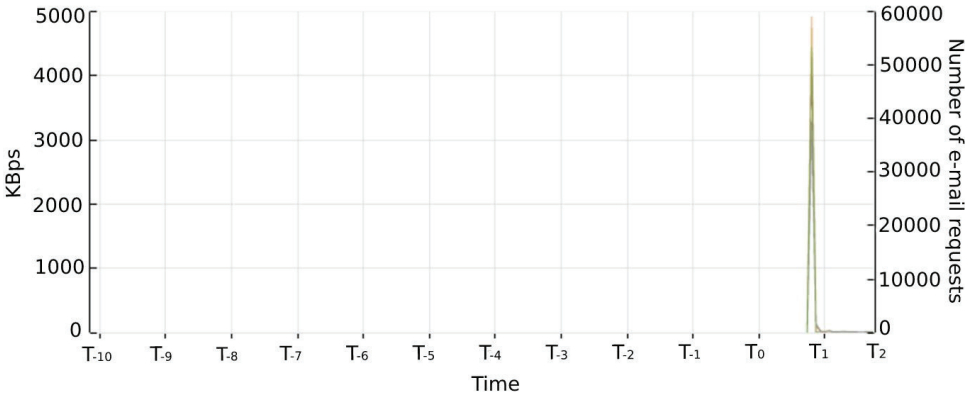

Fig. 6: Individual client network flow before and during the spam attack

loaded at any time of the attack, which leads to the conclusion that the implementation of the POW system on the SMTP server does not significantly affect the CPU usage, even during a distributed spam attack.

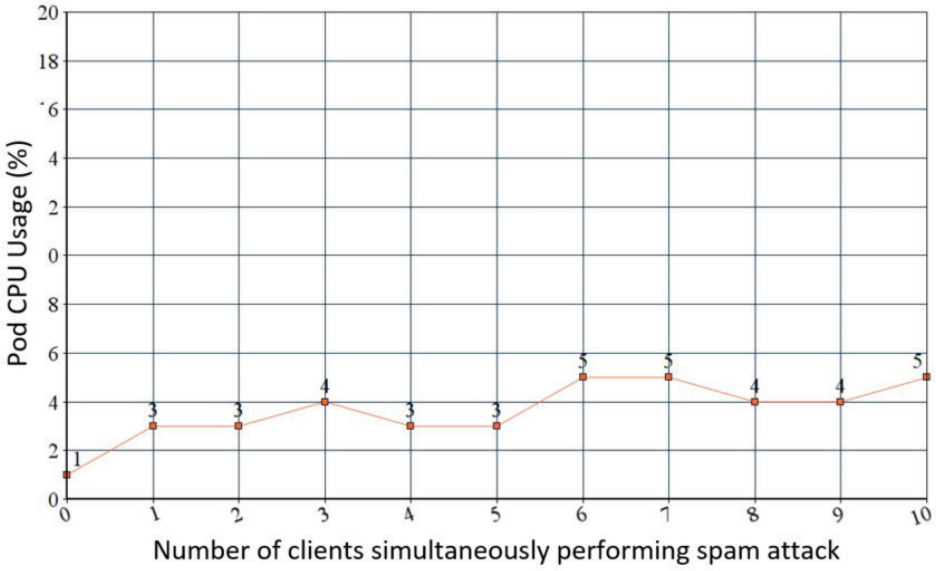

Fig. 7: Undistributed SMTP server CPU usage depending on the number of clients simultaneously performing spam attack

Unlike CPU load, spam attack has a significant impact on undistributed anti-spam SMTP server's memory load. A potential problem on the proposed anti-spam SMTP solution could occur when it is attacked by multiple clients, which can deplete its memory resources with a large number of parallel open connections (a problem close to DoS attacks). An example of one such attack is given in Fig. 8, which shows server memory usage as the number of requests from clients attacking single instance of the system increases over time. It can be seen that, after approximately 60,000 emails were sent, the server memory usage reached the upper limit. The memory usage shown refers to the load of the complete physical host executing the SMTP server. It can be seen that the anti-spam SMTP solution significantly loads the host's resources. At that point, the displayed server's load may result in a significant slowdown in request processing and an inability to respond to new client requests by rejecting their connections.

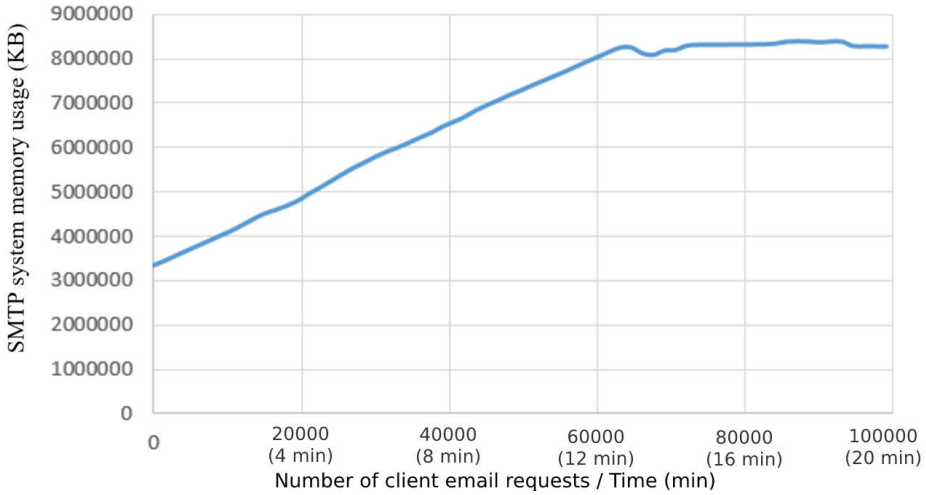

Fig. 8: Undistributed SMTP server memory usage during the spam attack

Previous analysis of the SMTP server memory load during a spam attack indicates a potential risk of congestion of proposed server resources by opening a large number of parallel connections. In order to address this problem, the server has been distributed and autoscaling on the Kubernetes architecture has been implemented. The next section will provide the analysis of the system memory usage during a spam attack, in the case where the anti-spam SMTP server is distributed on the Kubernetes architecture.

5.3 Autoscaling

In the configuration of the Kubernetes system initially there is one instance of the anti-spam SMTP server, which is not replicated until the system load indicates the need for a larger number of instances that will serve all of the client requests. Previously presented results of measuring performance of undistributed anti-spam SMTP server led to the conclusion that the parameter according to which the system should be scaled primarily is the memory usage, because the SMTP server CPU usage during a spam attack is not significant. The memory load in Fig. 8 is measured on the entire machine executing undistributed SMTP server (including OS and additional processes). Let us note that, since Kubernetes autoscaling is performed depending on the resource load of individual Pods, and since the execution of Pods, as a consequence of containerization, does not significantly affects the physical hosts' load, only the Pod's memory load is observed.

Table 2 presents performance analysis of the implemented system during the spam attack and demonstrates autoscaling the number of anti-spam SMTP server instances. The spam attack was simulated using 10 previously described virtual machines, where each of them is trying to send 10.000 of email requests. The number of SMTP clients which are participating in the attack has increased gradually. In order to analyze the impact of each added attacker, one new SMTP client was included in the attack every 120 seconds, as can be seen from the first two columns of Table 2. The third column gives the total memory usage of the Pods during the attack. At time T0, just before the start of the attack, it can be seen that the total Pod background memory usage is 12.9MB. As the number of clients increases, the total memory usage increases, approximately linearly. The fourth column gives the number of Pods (anti-spam SMTP server instances) that the Kubernetes system automatically lifts and maintains, depending on the average Pod memory usage given in the fifth column. The scaling of the number of Pods managed by the Kubernetes system is performed in the manner described previously. The upper limit of the average memory usage of the Pods running the anti-spam system is set to 100MB, in order to demonstrate autoscaling process in a given environment. Based on Table 2, it can be concluded that the system behaves according to the defined autoscaling configuration. Every time the average memory usage per Pod exceeds 100MB, the Kubernetes system creates another Pod (fourth and fifth columns of Table 2), protecting the anti-spam system from resource overload. At any time, the current number of Pods running the anti-spam SMTP system share the overall workload. In this way, the average memory

Table 2: Autoscaling anti-spam SMTP system instances during spam attack

Time (min)	# of clients / # of email requests	Total memory usage of the Pods (MB)	# of Pods	Average Pod memory usage (MB)
T0	0 / 0	12.9	1	12.9
T0 + 2	1 / 10000	36.6	1	36.6
T0 + 4	2 / 20000	52.03	1	52.03
T0 + 6	3 / 30000	78.2	1	78.2
T0 + 8	4 / 40000	111.9	1	111.9
T0 + 10	5 / 50000	142.4	2	71.2
T0 + 12	6 / 60000	170.8	2	85.4
T0 + 14	7 / 70000	195.4	2	97.7
T0 + 16	8 / 80000	218.4	2	109.2
T0 + 18	9 / 90000	237.9	3	79.3
T0 + 20	10 / 100000	259.5	3	86.5

usage per Pod is regulated to a value below 100MB. The data in the table are illustrated in Fig. 9.

This configuration of the system autoscaling prevents the possibility of its congestion during a spam attack, because the implementation allows the number of anti-spam SMTP instances to be automatically raised during the attack, which enables processing all of the client requests without overloading resources.

The proposed system provides a reliable, efficient and scalable anti-spam solution. System containerization enables its flexibility and portability, while execution in distributed Kubernetes environment provides easy scalability, reliability and fault tolerance, but also greater memory and processing power. Autoscaling of the system maximizes the use of available resources by maintaining the optimal number of anti-spam SMTP replicas, proportional to the current load. Additionally, not only will the operation of the anti-spam SMTP distributed system not be compromised by spam attacks, the proposed anti-spam system will slow down spam attacks, which will result in a significant reduction in the spam traffic amount on the network. In future work it would be useful to test how such a system affects legitimate mass mailing systems, which send a large number of emails that do not have spam content.

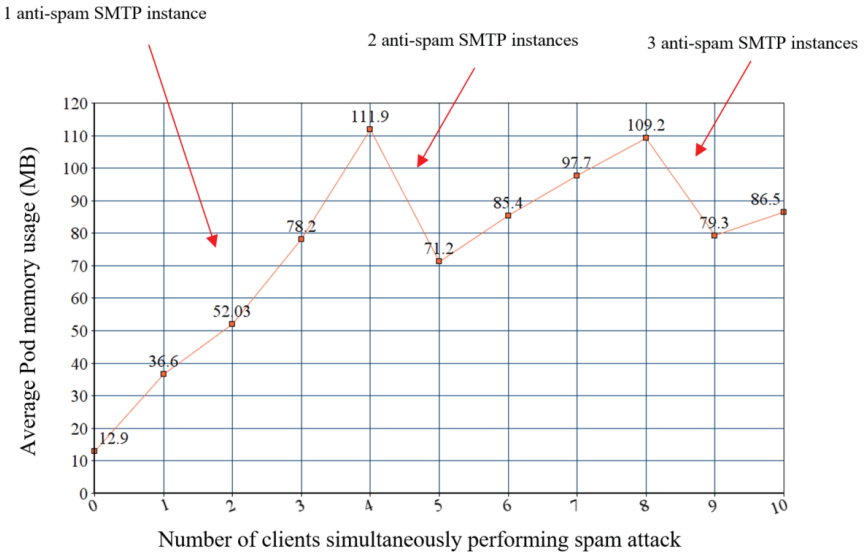

Fig. 9: Average Pod memory usage depending on the number of clients simultaneously performing spam attack

6 Conclusion

In this paper design and implementation of a distributed anti-spam SMTP system is proposed. The usage of PoW algorithm for verifying email client's credibility is analysed, and an extension of SMTP protocol, which enables requiring a certain amount of work from a sender, is proposed. Distributed Kubernetes architecture was used in order to prevent resource overload of the implemented system. The great benefit of the proposed system is the possibility of autoscaling the number of SMTP server replicas depending on the server load, which maximizes the use of available resources and makes the system resistant to spam attacks. The implemented system was evaluated during the distributed spam attempt, simulated by the custom-made traffic generator tool. Various performance tests have been given: (1) The proposed system's impact on the client's behaviour and the overall amount of spam traffic, (2) The undistributed anti-spam SMTP server's performance during the spam attempt, followed by a discussion about the reason for system distribution. Furthermore, (3) Autoscaling demonstration and distributed environment's performance evaluation during spam attack was given. We showed that the proposed solution has the possibility to significantly reduce

the amount of spam traffic, while processing tens of thousands of simultaneous SMTP client requests in a distributed environment and adjusting the proposed system's robustness.

Acknowledgments

References

[1] D. J. C. Klensin, "Email Statistics Report 2021-2025," 2021.

[2] H. Faris, A. M. Al-Zoubi, A. A. Heidari, I. Aljarah, M. Mafarja, M. A. Hassonah, and H. Fujita, "An intelligent system for spam detection and identification of the most relevant features based on evolutionary random weight networks," Information Fusion, vol. 48, pp. 67–83, 2019.

[3] H. Hu and G. Wang, "Revisiting email spoofing attacks," ArXiv, vol. abs/1801.00853, 2018.

[4] T. Wu, S. Wen, Y. Xiang, and W. Zhou, "Twitter spam detection: Survey of new approaches and comparative study," Computers & Security, vol. 76, pp. 265–284, 2018.

[5] A. M. Al-Zoubi, H. Faris, J. Alqatawna, and M. A. Hassonah, "Evolving support vector machines using whale optimization algorithm for spam profiles detection on online social networks in different lingual contexts," Knowledge-Based Systems, vol. 153, pp. 91–104, 2018.

[6] Z. Alom, B. Carminati, and E. Ferrari, "A deep learning model for twitter spam detection," Online Social Networks and Media, vol. 18, 2020.

[7] S. Kaddoura, O. Alfandi, and N. Dahmani, "A spam email detection mechanism for english language text emails using deep learning approach," 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 193–198, 2020.

[8] A. Karim, S. Azam, B. Shanmugam, K. Kannoorpatti, and M. Alazab, "A comprehensive survey for intelligent spam email detection," IEEE Access, vol. 7, pp. 168 261–168 295, 2019.

[9] S. Hameed, T. Kloht, and X. Fu, "Identity based email sender authentication for spam mitigation," in Eighth International Conference on Digital Information Management (ICDIM 2013), 2013, pp. 14–19.

[10] A. Schaub and D. Rossi, "Design and analysis of an improved bitmessage anti-spam mechanism," 09 2015, pp. 1–5.

[11] A. Biryukov and D. Khovratovich, "Equihash: Asymmetric proof-of-work based on the generalized birthday problem," Ledger, vol. 2, 2017.

[12] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake," IACR Cryptol. ePrint Arch., p. 452, 2014.

[13] D. J. C. Klensin, "Simple Mail Transfer Protocol," RFC 5321, Oct. 2008.

[14] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, ser. CRYPTO '92. Berlin, Heidelberg: Springer-Verlag, 1992, pp. 139–147.

[15] N. Poulton, Docker Deep Dive: Harness the full potential of your applications with Docker. Packt Publishing, 2020.

[16] M. Chae, H. Lee, and K. Lee, "A performance comparison of linux containers and virtual machines using docker and kvm," Cluster Comput, vol. 22, p. 17651775, 2019.

[17] K. Cochrane, J. S. Chelladhurai, and N. K. Khare, Docker Cookbook: Over 100 Practical and Insightful Recipes to Build Distributed Applications with Docker, 2nd Edition, 2nd ed. Packt Publishing, 2018.

[18] K. Matthias and S. P. Kane, "Docker: Up and running: Shipping reliable containers in production," 2015.

[19] M. Luksa, Kubernetes in Action. Manning Publications, 2018.

[20] N. Poulton and P. Joglekar, The Kubernetes Book. JJNP Consulting Limited, 2019.

[21] N. Gavrilovic and V. Ciric, "Design and evaluation of proof of work based anti-spam solution," in 2020 Zooming Innovation in Consumer Technologies Conference (ZINC), 2020, pp. 286–289.