ZINC

not another conference

# 2020 Zooming Innovation in Consumer Technologies Conference (ZINC)

Online, 26-27 May 2020

ISBN: 978-1-7281-8259-9

IEEE Catalog Number: CFP20ZIN-ART

www.GoZinc.org

# Design and Evaluation of Proof of Work Based Anti-Spam Solution

Nadja Gavrilovic
University of Nis, Faculty of Electronic Engineering
Nis, Serbia
nadja.gavrilovic@elfak.ni.ac.rs

Vladimir Ciric
University of Nis, Faculty of Electronic Engineering
Nis, Serbia
vladimir.ciric@elfak.ni.ac.rs

*Abstract*—**According to recent reports, most of the overall email traffic consists of spam, which represents an abuse for the purpose of mass distribution of unwanted messages. Spam can lead to serious attempts at a data breach or email consumer's identity theft, also. There are a lot of different anti-spam solutions. The aim of this paper is to design and evaluate an anti-spam solution based on a novel Proof of Work concept. The proposed solution requires a certain amount of work from a sender prior to the transfer of an email message. The extended SMTP protocol will be designed in order to enable the evaluation of client credibility using the Proof of work algorithm. The design of the proposed solution will be presented in detail. The implemented system will be evaluated in distributed spam attempt, simulated by Seagull tool. An evaluation of the client work, the impact on the overall amount of sent spam messages, as well as the impact of distributed spam attempt on the attacked server resource usage will be shown. We will show that the proposed solution helps in reducing spam traffic and server load, while it doesn't diminish the consumer experience of legitimate email users.**

***Keywords—proof of work, anti-spam, spam, email, smtp, pow***

## I. Introduction

A report published in [1] from March 2020 indicates that the number of globally sent email messages in a single day is as high as 306 billion, with a predicted growth of ~55 billion over the next four years. Approximately 50% of the total traffic is spam [1]. Spam represents the abuse of electronic systems and SMTP protocol for the purpose of sending mass unwanted messages. Usually these messages contain advertising content, but they can also contain viruses, various attacks on the consumer's computer, attempts at a data breach, etc. The basic feature of spam which makes it easy to use and very widespread is that sending a single email message does not represent a significant loss to the sender, either in terms of time, or financially. That is why sending a great number of messages of this kind is almost completely free.

The idea of pricing for every access to a shared resource, presented in [2], induced development of today's Proof of Work (PoW) systems. PoW introduces the concept of requiring a certain amount of client work prior to accessing a shared resource. In the case of fighting spam, that would require processing time prior to sending an email. Execution of cryptographic hash functions represents a way of investing sufficient processing time as proof of credibility. Recently, the use of PoW concept in various systems has been studied [3],[4],[5]. Implementation of PoW algorithm as an extension of SMTP protocol was discussed in [6].

The goal of this paper is to design an anti-spam solution based on a novel Proof of Work concept and evaluate the server load during distributed spam attempt. The proposed solution requires a certain amount of processing time from a potentially invalid sender, prior to the transfer of an email message. The implementation of such a system requires a SMTP protocol extension, which implies certain changes in client/server protocol defined communication. Protocol extension will be designed in order to enable the evaluation of client credibility using the Proof of work algorithm.

The design of the proposed solution will be presented in detail. The implemented system will be evaluated in distributed spam attempt, simulated by Seagull trafic generator tool. The distributed spam attempt will be performed by 10 nodes, thus simulating multiple clients and their simultaneous requests to the server. An evaluation of the client processing time, the impact on the overall amount of sent spam messages, as well as the resource usage of an SMTP server overloaded by requests of multiple clients will be shown. We will show that the proposed solution helps in reducing the spam traffic amount and server resource usage during the distributed spam attempt.

## II. Related Work

The initial idea of requiring a feasible amount of work when accessing a desired resource was first presented in [2]. In [7], a theory was presented that the PoW system of its own cannot be used to fight spam, due to its impact on legitimate email users. As a response to the findings pointed out in the aforementioned paper, [8] presents the idea of using reputation systems within the PoW system. Numerous papers were published which studied the implementation of this algorithm in various systems [3],[4],[5]. The common feature of all PoW based systems is the protection of access to shared resources. The PoW concept significantly gained in popularity when the concept of the Bitcoin cryptocurrency, which relies on the PoW system, was presented in 2008.

In [3] the authors suggested the use of the PoW concept in anti-spam solutions. They studied the implementation of a PoW anti-spam system on a Peer to Peer (P2P) network and evaluated its performance. The system from [3] differs from the system proposed in this paper in terms of how the amount of work required from the client is determined. In [3] the amount of work depends on the message itself, while in the proposed system it depends on the decision of the SMTP server. Unlike the P2P system, the proposed anti-spam solution is based on client/server communication.

## III. Background

In order to be able to design PoW based SMTP protocol, in this section we give a brief overview of basic SMTP protocol, as well as the basic concept of PoW systems.

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. The basic communication between the SMTP client and server during the successful transfer of a message is as follows [9]:

```
S: 220 smtpServer.example.com
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:<recipient@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, 03 March 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
```

In the communication shown above, **S** stands for the message that the server sends to the client, while **C** stands for the client message sent to the server. An email transaction begins with the command MAIL, which the client uses to define the sender email address. The second step is the definition of the recipient email address using the command RCPT, etc. In the beginning of every server message there is a three-digit numeric designation, which provides the client with information on the success of client action [9].

PoW systems emerged with the aim of preventing abuse of the processing power of a computer, for example during a Denial of Service (DoS) attack. The basic idea behind this concept is to request relatively small amount of processing time from a device which tries to access protected resource. This prevents the activity of the basic feature of the aforementioned attacks – a large number of access attempts at a resource over a short period of time [2].

The principle behind how a PoW system works is based on a typical cryptographic scenario, during which one side that takes part in the communication attempts to prove its validity to the other. Specifically, the client who is requesting a service or resource from the server should first prove its reliability. This proof is realized through a certain amount of processing time, with the aim of fulfilling the criterion issued by the server [2].

Most often, carrying out mathematical or cryptographic functions represents a way of investing sufficient processing time as proof of credibility. 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 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. Then, the client sends that solution to the server. The task of the server is to check the validity of the obtained solution and identify the client as reliable or unreliable, based on the previous analysis [4].

## IV. System Overview

The idea proposed in this paper is the possibility of asking a potentially invalid client for proof of credibility by requiring certain amounts of processing time. In this way, without impacting valid email users, the server can conclude whether the client it is communicating with has the intention of abusing the server. The implementation of such a system requires certain changes in the implementation of the SMTP protocol.

At the beginning of the communication, the 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 changes enable the server to send integer value, which represents the *weight*. Its value determines the criterion which the client has to meet. By executing a cryptographic hash function on the email message, the client must generate a sequence which has as many zeroes in the beginning, as were defined by the previously received weight parameter. By changing the value of the weight parameter, the server could require various amounts of processing work from various clients.

The execution of the same hash function on the same data sequence always results in the same output. That is why we request from the client in the extended SMTP protocol to append a *nonce* value to the email data, and then to calculate the hash value of the whole sequence. The only way to determine a nonce value which 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 increments. Bearing in mind that the feature of a hash function is to provide drastic changes in the output for small changes in the input, completely different values are obtained with each execution of the function on the data.A single execution of a hash algorithm on the data does not require significant processing work, but obtaining a satisfactory output sequence is sufficiently rare to enable the overall process to take the client significant processing time.

Once the nonce value used to satisfy the issued requirement is found, the client forwards it to the server. After receiving the nonce value from the client, server checks it to ensure if the obtained value meets the set requirement, that is, whether the required amount of work has been put in. The check is achieved on the server side through a single hash function execution on the data sequence which consists of previously obtained email message and recently received nonce value appended to the email. This signals that a significantly greater amount of CPU time is required to solve the given problem on the client than the amount required to verify the solution on the server. If the server determines that the applied nonce value meets the set requirement, the client is characterized as valid and email is successfully transferred. However, if the nonce is not verified by the server, server notifies the client that the email transfer has failed.

A valid consumer, unlike a client using spam, rarely sends the server great many email requests over a short period of time. Thus, even if the valid email consumer has been incorrectly evaluated by a reputation system as potentially dangerous, the processing time needed for an adequate summary of a small number of emails will not render the use of the email service more difficult. Thus, valid consumers' use of the server has not undergone any noticeable changes. The complete client/server communication with the proposed changes, is shown in the following:

**S**: 220 smtpServer.example.com
**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:<recipient@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, 03 March 2020 16:02:43
**C**: Subject: Hello
**C**: Hello world!
**C**: .
**S**: 250 Challenge number 03
**C**: Nonce : 79745
**S**: 250 Queued mail for delivery
**C**: QUIT
**S**: 221 Server closing connection

The emphasized part of the communication shows the validation of the consumer. The server requires processing time from the client as a potentially dangerous one. The value of the parameter *Challenge number* 3 denotes that the client must generate an output sequence which in the beginning had precisely 3 zeroes. As previously described, the client through multiple executions of the hash function on the email message and the nonce value appended to it, attempts to generate an output sequence which will satisfy the set requirement. In the given example, the value of the *nonce* parameter is 79745. It denotes that the email message, with the value of 79745 appended to it, composed the right sequence which met the requirement given by the server.

## V. PERFORMANCE MEASUREMENT AND ANALYSIS

The SMTP server with a PoW extension was implemented using .NET Framework. The server was tested on an AMD A10-5745M model processor, with a clock speed of 2.10 GHz. Distributed spam attempt was simulated by a cluster of 10 SMTP clients, using the Seagull tool on each of them.

### A. An evaluation of the client work

The amount of processing time required from a client has been evaluated through multiple testing of various weight values assigned by the server. The results presented in Table I were obtained. With an increase in the values of the weight parameter, the amount of required CPU time increases, and can be significant (Table I). The precise CPU time varies depending on the speed of the client processor, its usage and the probability function, which determines the precise step during which the desired value will be obtained. The value of the standard deviation is significant for each of the weight values, and the reason are the great deviations in the amount of client work, precisely because of the previously discussed probability function.

TABLE I. AN EVALUATION OF THE CLIENT WORK FOR VARIOUS WEIGHTS

| Weight | Average time needed to perform the function | Standard deviation of the performance time | Average number of performed hash functions | Number 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 |

### B. The impact on the amount of sent spam messages

Hash Power (HP) parameter represents the average number of executions of hash function per second, that a CPU can perform. The testing has shown that the number of client hash function executions per second per individual request decreases from HP, with an increase in the number of simultaneously sent requests from a single client due to its CPU overload. Testing results can be seen in Fig. 1. A direct consequence is the increase in the duration of the individual client/server connection, during which the client CPU is overloaded. How long each individual request will last depends on the number of simultaneously opened connections, as shown in Fig. 2.

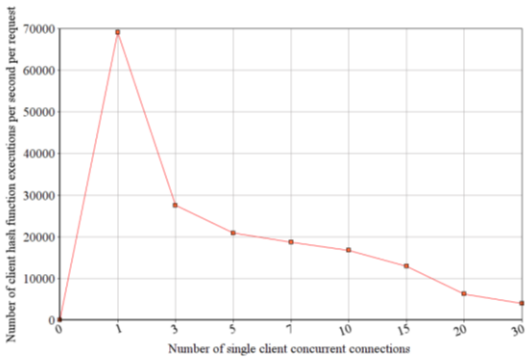

Fig. 1. The number of client hash function executions per second per individual request depending on the number of simultaneously sent requests.

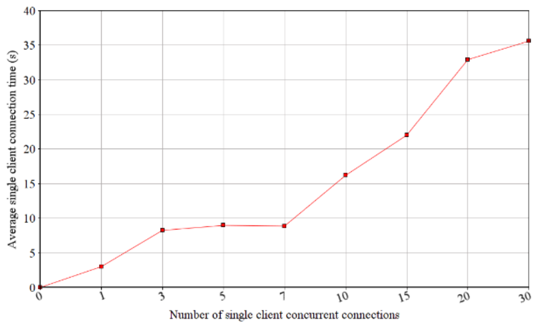

Fig. 2. The average duration of a client-server connection depending on the number of simultaneously sent single client requests.

A consequence of the displayed results is that the number of possible client outbound emails 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 HP parameter, as

$$N_e \approx HP / H(T), \quad (1)$$

where $N_e$ is the number of outbound messages from the client per second, *HP* is the hash power of the client, *T* is weight parameter, and *H(T)* is the average number of required executions of the hash algorithm for the weight *T*.

For a weight value of 3, the average number of executed hash functions shown in table I and the HP of the aforementioned processor, according to (1) the number of sent client messages per second is 0.345. By testing the implemented system for various numbers of client requests per second, results were obtained and are shown in Fig. 3.

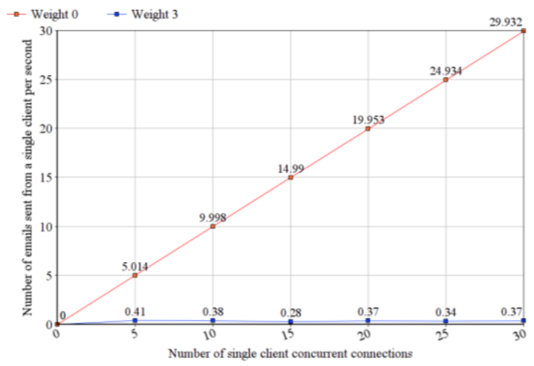

Fig. 3. The number of emails sent from a single client per second, depending on the number of sent client requests, without and with CPU work requested

Based on the presented results, 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, irrespective of the number of requests sent. Using the weight value of 3 limits the speed of sending spam messages, which reduces the number of messages of that kind on the network. In the case where CPU time is not requested from the client, that is, where the value of the weight is 0, the client sends approximately the same number of emails as requests sent to the server, and the email is sent almost instantaneously.

### *C. The evaluation of email server resource usage during distributed spam attack*

By testing the client CPU usage depending on the number of simultaneously sent requests for sending an email, while using weight value of 3, the results presented in Fig. 4 were obtained. The sequential execution of hash functions overloaded the tested client CPU to 100% in the case of 30 simultaneously sent emails.

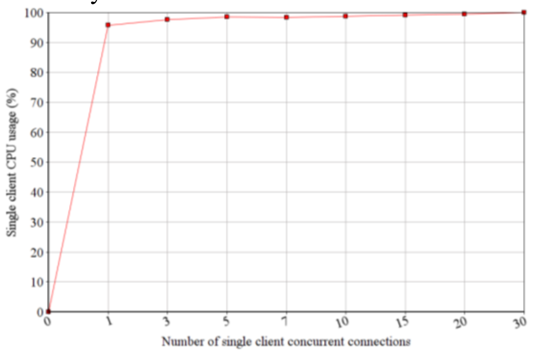

Fig. 4. Client processor usage depending on the number of simultaneously sent requests.

The implemented SMTP server load was tested during the distributed spam attempt, simulated by a cluster of 10 clients. Server load was evaluated by varying the number of clients sending requests at the same time. Each computer used the Seagull tool to simultaneously send 30 email requests, which the server accepted and processed. With 30 opened connections with the server, the CPU overload for all the clients was 100%. The average CPU usage of the SMTP server during the first 180 seconds after accepting all the connections was monitored. The result is shown in Fig. 5.

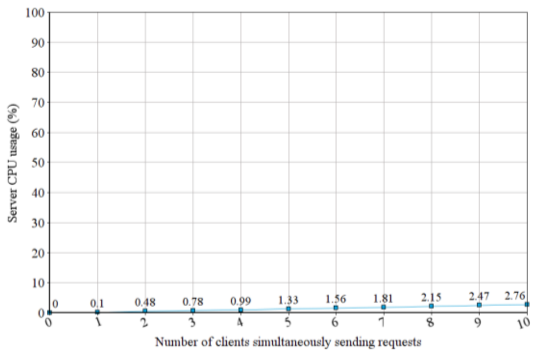

Fig. 5. Server processor usage depending on the number of clients simultaneously sending requests, with each client processor usage of 100%.

We can conclude that the CPU usage of the implemented server was not greatly affected by the distributed spam attempt. Also, by limiting the speed of sending spam messages, proposed solution affects the most pronounced feature of spam – sending out a large number of email messages over a short period of time. By reducing overall spam traffic and conserving server load, consumer technologies which use email services (e.g. mobile mail clients and applications) can be used with less disturbance and security threats for valid consumers.

## VI. Conclusion

In this paper we designed and evaluated an anti-spam solution based on a novel PoW concept. In order to enable the evaluation of client credibility using the PoW algorithm, the SMTP protocol extension has been designed. The proposed system has been evaluated in distributed spam attempt, simulated by Seagull tool. An evaluation of the client work, the impact on the overall amount of sent spam messages, as well as the impact of distributed spam attempt on the attacked server load are shown. It is shown that the proposed solution helps in reducing spam traffic and server load, while it doesn’t diminish the consumer experience of legitimate email users.

## Acknowledgment

This work was supported by the Serbian Ministry of Education, Science and Technological Development [grant number TR32012].

## References

[1] The Radicati Group Inc, Email Statistics Report, 2020-2024.

[2] C. Dwork, M. Naor, "Pricing via processing or combating junk mail", In Proc. Of the 12th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag, Berlin, 1992, pp. 139–147.

[3] A. Schaub, D. Rossi, "Design and analysis of an improved bitmessage anti-spam mechanism," 2015 IEEE International Conference on Peer-to-Peer Computing (P2P), Boston, MA, 2015, pp. 1-5.

[4] A. Biryukov, D. Khovratovich, Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem, 2016.

[5] I. Bentov, Ch. Lee, A. Mizrahi, M. Rosenfeld, Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake, y. SIGMETRICS Perform. Eval. Rev. 42, 2014, pp. 34–37.

[6] N. Gavrilović, "Design and implementation of SMTP network protocol extension for proof of client’s work", Proc. on IEEESTEC 12th student projects conference, Faculty of Electronic Engineering, Niš, 2019, pp. 321-324.

[7] B. Laurie, R. Clayton, "Proof-of-work proves not to work", Workshop on Economics and Information Security, 2004.

[8] D. Liu, "Proof of work can work", Fifth Workshop on the Economics of Information Security, Indiana University, 2006.

[9] J. Klensin, "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008