KTH Computer Science
and Communication

RB-PoW: A reputation based approach to cryptographic mitigation of denial-of-service attacks

FREDRIK LILKAER, CHRISTOPHER TELJSTEDT

Bachelor of Science Thesis
Supervisor: Douglas Wikström
Examiner: Mårten Björkman

Abstract

Denial of Service (DoS) aims to drain the underlying resources of a service and is a increasing problem for today's service providers. In this paper, we propose a novel reputation based proof-of-work (RB-PoW) protocol based on adaptive scaling of puzzle difficulties to mitigate various denial-of-service attacks. A testing framework was developed to examine RB-PoW's effectiveness against classical PoW protocols in simulated DoS attacks. We concluded that in flooding type of denial-of-service attacks the RB-PoW model is superior to the classical PoW that is based solely on server load. Both offer similar performance when exposed to drain-type denial-of-service attacks.

Referat

Ändamålet med Denial of Service(DoS) attacker är att dränera en tjänsts underliggande resurser. Attacktypen är ett växande problem för dagens tjänsteförmedlare. I detta kandidatexamensarbete föreslår vi en ny ryktesbaserad Proof of Work (RB-PoW)protokoll som bygger på anpassningsbar skalning av pusselsvårigheter för att dämpa diverse denial-of-service-attacker. Ett testramverk utvecklades för att undersöka RB-PoWs effektivitet i jämförelse med den klassiska implementationen av PoW protokoll i en simulerad DoS-attack. Våra slutsatser är att RB-PoW är överlägsen i översvämmningsattacker i jämförelse med den klassiska PoW varianten som är enbart är baserad på serverbelastning. Modellerna har likvärdig prestanda när de blir exponerade för dräneringsattacker.

Statement of Collaboration

Fredrik initiated the programming work build the basis of a framework for developement. The protocol was decided in collaboration while Fredrik implemented the protocol, general server structure and a “normal service”- interface. Christopher continued the work with an web-based simulation interface, able to control an attack behaviour against the site. A monitor page was developed by Fredrik and then improved by Christopher. Further improvements where developed in conjunction as need arose.

Christopher created the boilerplate of the document and wrote the initial introduction. Both participated equally in respect to writing, analysis of results while Christopher did the final editing. The thesis is a collaborative effort.

Contents

Contents

1 Introduction

The Internet continues to grow and has since a time back enabled a single source to be connected to several millions of geographically dispersed computers. Security flaws of a new magnitude has as a consequence been introduced; enabling a single computer to be attacked by millions of sources at once.

This threat also known as denial-of-service (DoS) attacks is a growing concern as these attacks have shown to disable even well-known services. Unlike other attacks the primary goal of denial-of-service attacks is to restrict or disrupt the availability of the service to its legitimate users. It is essentially a targeted effort to prevent a service from functioning properly by draining the underlying computer resources. One attempt to counter DoS attacks and to improve service survivability is a computational approach called Proof of Work (PoW). Proof-of-work also known as *Client Puzzles*[1, 2] is cryptographic in flavor and requires the clients to solve an instance of a predefined problem in order to gain access to the server's resources. Proof of Work, when initially proposed, seemed like a very promising method to fighting DoS attacks but since Laurie and Clayton published the paper *"Proof of Work" proves not to work* in 2004, many PoW schemes were disregarded. However, there was an aspect that Laurie and Clayton did not consider, puzzle schemes that are adaptive[4, 5].

In this paper, we examine the use of adaptive scaling of proof-of-work problems to mitigate various DoS attacks. We propose a new Reputation Based Proof of Work protocol(RB-PoW) that scale problem difficulties based on request rate behaviour of a user.

The organisation of this paper is as follows. Section 2 explains the theory behind the reputation based proof-of-work protocol and assumptions that has to be made as well as notation of the proposed protocol. A brief view on the software testing framework is given in section 4. An outline of important system parameters and a detailed explanation of the simulation experiments are given in Section 4. The results of the simulation experiments are presented in section 5 and a concluding discussion is provided in section 6.

Proof of Work and Related Work

The fundamental property of Proof of Work is that its *puzzles* are moderately hard to solve but easy to verify. This concept was originally proposed by Dwork and Naor 1992 in "Pricing via Processing or Combatting Junk Mail" as a way to counter e-mail spam by increasing the costs of sending spam, thus making e-mail spam economically infeasible. Dwork and Naor originally called this a *pricing function* since it effectively puts a price on using a service.

The concept of proof-of-work was reinvented by Back, who later proposed "hash-cash" as a DoS preventing method in *Hashcash - A Denial of Service Counter-Measure*. Back's idea was to require the requester of a service to compute a cryptographic hash in which the most significant bits start with a certain number of zeroes.[1] Consequently, the average work to solve a ”hashcash” puzzle is exponential to the number of zero bits required vis-à-vis computing a single hash to verify[8].

The idea of proof-of-work has since then been adapted into various field such as mitigating denial-of-service attacks [9], limiting the rate of new TCP connections [2] and as an inducement in peer-to-peer networks[8, 10]. Variants of proof-of-work schemes are further discussed in “Resource inflation threats to denial of service countermeasures”.

Problem definition

Proof of Work has been shown to potentially work as a prevention mechanism to at least mitigate the effects of a DoS attack[2]. However, scaling problem difficulty with server load may render problem difficulties that induces a significant disruption to the legitimate client. Naive proof of work implementations similar to hashcash are significantly CPU-bound[11]. Consequently, a service protected by trivial PoW schemes heavily penalises users of lesser hardware, as supported by our findings presented in table 5.1, as well as Tsang and Smith[12].

Problem statement

- Can an individually adaptive Proof of Work scheme be optimised to penalise malicious behaviour while still enabling legitimate clients to access the service, thus improving performance versus a conventional load scaling system?
- Is there a viable way to implement a Proof of Work system that maintains service, even to mobile users, while the system is under attack?

Purpose and method

The purpose of this study is to research ways to improve the classical Proof of Work in such a way that legitimate users are less affected by the Proof of Work than the participants of a DoS attack. Furthermore to find a way to dynamically scale the required proof of work when dealing with different hardware.

A PoW difficulty scaling model is developed and its performance is measured against a naive CPU load difficulty scaler. To support the testing, a Proof of Work testing framework is designed and implemented.

Scope and delimitations

This study explores a reputation based approach to scaling proof-of-work puzzle difficulties, thereby covering necessary theory of puzzle-based mechanisms specifically hash-reversal based puzzles. Furthermore, the study aims to cover explanations of the implemented protocol, simulation experiments and results of significance.

[1]In the first scheme of *hashcash* the initial bits of two hashes had to match - what mentioned in the text is actually a recent improvement.

There is various types of proof-of-work protocols as of recently. Hence, the study does not specifically aim to cover these but might be mentioned throughout the paper. Various types of denial-of-service attacks exists, some which aim to disable lower level layers. This paper is limited to DoS attacks on application level layer by technical reasons. Otherwise, a proof-of-work scheme can be circumvented by an attack on a lower abstraction layer.[2]

One important parameter of any DoS mitigation scheme is the time to live (TTL) parameter of an initiated but idle connection. The study does not cover TTL optimisation. The variable is fixed as infinite throughout this paper.

Terminology

A brief explanation of terminology and abbreviations used throughout this paper follows:

- **Client / User**: *Client* and *user* will be used interchangeably depending on the context where the term client tend to be more software oriented in contrast to user which usually is referred from an “in system” perspective.
- **Adversary**: Attackers or users with malicious intent will be referred to as *adversaries*. An adversary is technically a client in the system, but will not be regarded as such throughout this paper, since the intentions of the adversary is to exploit and/or interrupt the service in contrast to the legitimate client.
- **SHA2**: refers to the SHA256 secure hash algorithm[13].
- **Puzzle**: A problem instantiation of the problem type specified in the protocol. See section 2.1.2.
- **Problem**: A general problem, may also refer to a Puzzle.
- **PoW**: Abbreviation for Proof of Work.
- **DoS**: Abbreviation for Denial of Service (attack).
- **RB-PoW**: Our reputation based adaptive Proof of Work scheme.

2 Adapting Proof of Work

Denial of Service attacks are often possible due to the low price of a request, for an adversary often a single network packet[5], compared to the work the service provider performs. Puzzle-based proof-of-work limits an attacker’s possibilities to impose a high work load on a server from a sole client by increasing the cost of each request. Laurie and Clayton argued that Proof of Work, assuming a fixed cost proof-of-work pricing function, would not work. In this section we present an

[2]Application layer and abstraction layer of the OSI model.

alternate interpretation of proof-of-work incorporating problem scaling and individual adaptation, that may in fact be a practically viable scheme to considerably attenuate the effects of a denial-of-service attack.

The RB-PoW scheme utilises a hash-reversal puzzle very similar to hashcash. The difference to hashcash is that RB-PoW uses SHA2 in place of SHA1, since it is a cryptographically stronger hash[14]. The puzzle is a seed $P_i[j]$[3]. The client needs to find an $S_i[j]$ such that the computed hash $h = H(S_i[j]||P_i[j])$ holds the property that the leading d (issued difficulty) digits in the hex representation of h are all equal to zero. It is computationally infeasible to find x for a given h such that $h = \text{sha2}(x)$[13], consequently the only way to find $S_i[i]$ is by sequential trial. For $d_{l+1} = d_l + 1$, only $\frac{1}{16}$ of solutions accepted for difficulty d_l are accepted. The complexity of finding $S_i[j]$ is thus $O(16^d)$. This is however an amortized complexity, since one can be lucky and find a solution in the first hash test, or be forced to seek nearly the complete solution space. The actual running time is in fact geometrically distributed with an expected outcome of 16^d trials before finding the solution, but the running time may be improved to approach a normal distribution with the introduction of sub-puzzles[15]. The reputation based proof-of-work model uses subpuzzles to normalise expected solving times but also to scale problem difficulties. To harvest the almost normalised run times we use a minimum of 16 sub-puzzles, as suggested by Henriques and Nordmark. A (sub-)puzzle set may have a cardinality as large as 256 to enable integer scaling of problem difficulty. The central difficulty scaling model of RB-PoW will be further addressed in section 2.5.

2.1 RB-PoW Protocol

2.1.1 Protocol notation

To formalize the Proof of Work protocol a few notations will be introduced. Consider the following notations:

$\mathbf{M}_i$ = to be i^{th} execution of the protocol M by either a legitimate user or an adversary.
h = a puzzle generator function.
$\mathbf{P}[j]$ = the j^{th} sub-puzzle in P.
m = the number of sub-puzzles, this equals to the size of the set P.
d = an integer indicating the difficulty of the problem set P.
$\mathbf{S}[j]$ = the j^{th} solution in S and a solution to j^{th} sub-puzzle in P.
z = a function that return the number of most significant bytes that is zero.

2.1.2 Protocol description

Now let us describe the details of our proposed reputation based puzzle protocol between client and server. Prior to initiating protocol **M** the client C_i starts by requesting(R_i) a service from the server. The server responds with a packaged set

[3]For a precise definition of $P_i[j]$, i and j, please see section 2.1.1.

of sub-puzzles P_i of size m and a difficulty d. Each sub-puzzle $P[1] \dots P[m]$ is a seed derived by $g(t,x)$ where x is value generated uniformly-at-random.

The client C_i must solve each sub-puzzle $P_i[j]$ of P_i by finding a value $S_i[j]$ so that the computed hash $H(S_i[j] \mid\mid P_i[j])$ has at least d leading zeroes. If such a hash is found then the solution $S[j]$ is a solution to the sub-puzzle $P[j]$.

The server then verifies each solution of S_i by computing $H(S_i[j] \mid\mid P_i[j])$ with a stored copy of $P_i[j]$ and confirms that the solution has d leading zeroes. If all solutions are correctly computed the client C_i's request R_i will be granted.

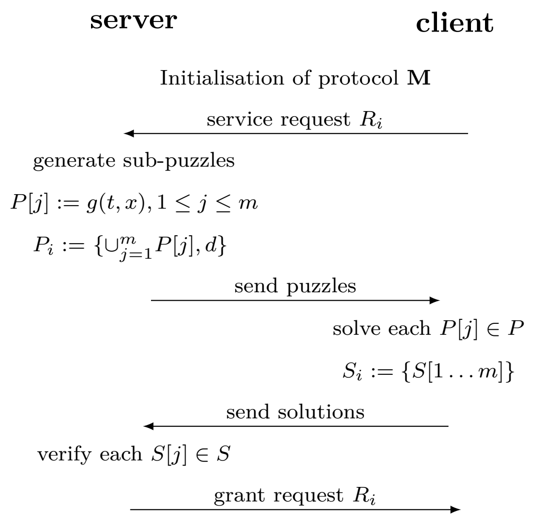

Figure 2.1. Diagram of Proof of Work protocol

If the reputation system deems the server to not be under attack i.e. under normal server operation, the problem package will be the empty set, indicating that no puzzles are being distributed. Hence, the client will "solve" this empty set with no effort and then respond back to the server.[4] The server will then verify the solution for P_i and grant access to R_i of protocol $\mathbf{M}$.

If the reputation system deems the server to be under attack the difficulty of the problem set for request R_i will be decided by the historical and current behaviour of client C_i.

The following sections will present server-side metrics needed to define client behaviour and explain assumptions made for the test environment. We will precisely define the meaning of client behaviour in our model and finally carry on to develop and describe the reputation based proof-of-work model.

[4]This design decision may be questioned, arguing that the handshaking process with empty sets could bring unnecessary load on the server. However, empty sets will only be distributed when server load is low thus the extra verification can be afforded.

2.2 Assumptions

2.2.1 Attack model

In this section, the attack model of the simulation experiment will be described. In order to specify an attack model and to simulate an attack, some assumptions regarding the adversary are made. Assumptions enable us to test a DoS mitigation scheme in the application layer; if any assumption does not hold the protection can be circumvented by an attack through a lower abstraction layer[5]. We assume a server *Server*, a network with clients $\{C_i\}$, and adversary *Ad*.

Assumption 1 *Ad* cannot modify any packets sent between C_i and *Server*.

This is necessary to assume, in particular that it cannot modify legitimate users packets. If Assumption 1 does not hold and packets can be modified then a denial-of-service attack can be initiated simply by corrupting all packages thus rendering any PoW protocol ineffective.

Assumption 2 *Ad* cannot delay any packets sent between C_i and *Server*.

By similar reasons as Assumption 1 we must assume that the adversary cannot delay other clients' packets. If Assumption 2 does not hold and an adversary can delay packets arbitrarily then the she can mount a denial-of-service attack without draining the resources of the server itself.

Assumption 3 *Ad* cannot oversaturate the transfer layer[6] or network layer[7] of the *Server*.

An adversary must be able to perform a denial-of-service attack by sending large amount of requests to the server. However, we must also assume that the adversary cannot bring down the server by simply over-saturating the server ports or network with sheer volume of the adversary's requests.

2.2.2 Environment

Assumption 4 We will assume that every action the server performs have unit cost, the assumption does not lessen generality, since an external difficulty scaler could be added to each action the server performs.

[5]Abstraction layer in the OSI model
[6]Transfer layer of the OSI model
[7]Network layer of the OSI model

Assumption 5 We will assume that the server is dimensioned to handle load under normal operation.

2.3 Metrics

In order to establish a quantitative description of metrics required by the difficulty scaling model as well as simulation experiments this section will outline three sets of parameters referred to as quantifiers, observables and controllables.

2.3.1 Attack Quantifiers

Service time reflects measurement in the quality of a requested service by a client. Assuming that the service is web browsing the service time could be the time until a web page is completely transfered. If an adversary aims to mount a DoS attack the service quality during the attack can be measured by the average service latency for legitimate users.

2.3.2 Server Observables

The difficulty of PoW problems handed out as a response to requests are based a set of selected parameters that in different ways reflects the server's resource consumption.

Number of established connections: keeps track of how many clients[8] that is currently connected to the server.

CPU load averages: tell us whether the physical CPU utilization is over or under saturated. A perfect utilization is when the CPU is busy but no process is stalled. In general load average differ from CPU usage in two significant ways:

1. the CPU usage measures the instantaneous snapshot while load averages measure the trend in CPU utilization.
2. the CPU usage only measures how much was active during measured time-frame while load averages take all demands for CPU into account.

If the server has four CPUs running and the reported load average is 4.00 then the CPUs are perfectly utilized[16].

Request rate is the quantified amount of requests for a service that a single *client* or a *group of clients* asks for. This measurement is slightly modified to measure the average time between requests to show the amount servicing required by each client and the average of all clients in a given time-frame. By introducing a rating scheme that weights the individual requesting rate of a client with the average request rate of all clients the Reputation System can hand out problems scaled to the behaviour of each client. Thereby limiting the amount of service to a client with malicious behaviour and furthermore giving a natural limit to a DoS attack.

[8]This is actually the number of web sockets open on the server side.

2.3.3 Server Controllables

The server resources is managed by the reputation behaviour model, generally it has two settings that can be regulated. Firstly, *Puzzle Difficulty*, the difficulty of a puzzle is based on parameters in Server Observables. However, the scaling of puzzles can be tuned to respond more harshly or forgiving based on the parameters.

Secondly, *Connection Time-out* is used to control the lifetime a connection. This could be a finite duration meaning that if no solution is sent back to the server within the connection lifetime the connection will be closed. However, in the scope of the simulation experiments this duration is set to infinite.

2.4 Behaviour model

An essential principle of RB-PoW is the RB-PoW definition of fairness:

Every client has equal rights to the server's resources.

Figure 2.2. RB-PoW Reputation Model Statement

The behaviour model does not reason about users as legitimate or malicious in an absolute meaning, it regards each user as a potential evil-doer with varying maliciousness. A "good" user is thus a user that is less taxing on the system whereas an adversary is a user that is more taxing on the system than the global average.

Assumption 4 implies that the demands of a client can be measured by the number of connections concerned initiates in a given time-frame, or the inverse: how long time is passed between each request. To quantify the behaviour of a user we measure average time between request instead of average request rate per time unit. Behaviour b_i is calculated as an exponential moving average and is recalculated at every request the client performs:

$$b_i = \alpha \cdot \delta + (1 - \alpha) b_{i-1}$$

where δ is the time difference since the last request. At every request, the global average behaviour B_i is also recalculated:

$$B_i = \beta \cdot \Delta + (1 - \beta) B_{i-1}$$

where $\Delta = \delta \cdot n$, n is the number of users currently connected to the system. The multiplication averages the time between request and enables us to compare the value to local values. One advantage of measuring the time between request versus counting requests during a sampling period is that it can be implemented more efficient. An even more interesting property is the effect on behaviour change rate. Sampling number of request during a period and averaging over n last periods means that a client is considered “good” again after previously considered “bad” just as fast (in time) as the opposite if concerned client changes behaviour pattern. Averaging over the last n requests instead will cause the system to respond faster (in time) if a client start acting maliciously (suddenly increasing request rate) and demanding a longer time for the client to redeem her behaviour.

2.5 Reputation Mechanism as a Difficulty Scaler

The RB-PoW Behaviour model is essential to the RB-PoW adaptive denial-of-service mitigation scheme. The behaviour model provides a metric describing the individual client’s load on the service, as well as the global average load per client. With the introduction of the behaviour model, the puzzle difficulty scaling model becomes simple to describe:

- The client C_i behaves in a manner comparable to B_i
  - The request R_i is assigned a problem difficulty suitable to server load.

- The client C_i demands less resources compared to the general behaviour ($b_i > B_i$).
  - The request R_i is assigned a problem difficulty easier than the general difficulty.

- The client C_i demands more resources compared to the general behaviour ($b_i < B_i$).
  - The request R_i is assigned a problem difficulty harder than the general difficulty.

Assumption 5 implies that the DoS protection functionality is unlikely to be activated during normal operation. It also implies that if the server is under attack, the average adversary is likely more taxing on the system than the average legitimate user, unless the offender has access to orders of magnitude more adversaries than the normal user base. The RB-PoW model leverages these assumptions to maintain service to legitimate desktop and mobile users: if the number of attackers is low in compared to number of legitimate clients the global average will represent the legitimate user base. In this case the adversaries will have a faster request rate than the average population and will be alotted harder problems. If the number of attackers are more than the number of legitimate users, the average will instead represent the adversaries. If legitimate users access the service less frequently they will receive easier problems compared to the problems received by the opposers.

The RB-PoW model further defines a client’s relative behaviour as $b_{r_i} = \frac{b_i}{B_i}$, which is used as a difficulty multiplier. The RB-PoW is a conceptual model that should be adapted to the specific application. Our implementation, which contains more than a few arbitrary constants, follows:

```
func rp_scale_model(p Param) Difficulty {
        if math.Max(p.Cpu.Load, p.Cpu.Avg) < cpu_thres {
                return ZeroDifficulty
        }
        if p.Local.LongMean > 2*max(p.Global.ShortMean, p.Global.LongMean) {
                if p.Local.ShortMean > 3*p.Global.LongMean
                        && math.Max(p.Cpu.Load, p.Cpu.Avg) < cpu_thres+20 {
                        return ZeroDifficulty
                }
                return BaseDifficulty
        }
        diff := BaseDifficulty.multiply(1 + int((math.Max(p.Cpu.Avg, cpu_thres) - cpu_thres)))
        return *diff.multiply(1 + int(5*p.Global.LongMean/p.Local.ShortMean))
}
```

Difficulty is described as number of leading zeroes required in each subpuzzle, and the number of subpuzzles.

```
type Difficulty struct {
        Zeroes   int
        Problems int
}
```

Multiplication of a difficulty with an integer is defined as multiplication with number of subpuzzles. The number of subpuzzles is divided by 16 as the number of zeroes is incremented by one to not generate excessive amounts of easy subpuzzles. 16 puzzles is the minimum amount of subpuzzles to normalise the solving time for a particular difficulty[15].

```
func (d *Difficulty) multiply(f int) *Difficulty {
        r := *d
        r.Problems *= f
        for r.Problems > 256 {
                r.Problems /= 16
                r.Zeroes++
        }
        return &r
}
```

3 System Architecture

Since one requirement on our Proof of Work system is that it minimises differences between a wide variety of devices; it needs to support both desktop and laptops with different operating systems as well as cellphones and tablets. In order to minimise the development effort and maximise maintainability of the code base, a multi platform solution was sought for web-based simulation interface.

A web-based solution makes the application portable, but the web is not inherently stateful thus an emulation of a general service is limited. However, the advent of web-sockets enables a truly multi-platform proof-of-work client in html and javascript while maintaining the generality and plasticity of a natively written socket based application.

The server side was implemented in Google's novel programming language golang[17]. The real strengths of golang in this context is actually not performance nor simplicity[9] but rather it's standard libraries, which in other contexts may appear immature. Golang includes standard libraries for both html template generation, http web serving as well as web-sockets. This package makes for an ideal platform for an application that needs to deliver the client application[10] to potential clients as well as servicing client requests in the model application reachable through the web-socket interface.

Communication is performed over a web-socket per client, with a single message type which is (de-)serialised to (from) JSON. JSON is natively supported in the Javascript client, and the golang web-socket library supports JSON (de-)serialisation. Thus no byte parsing specific to our protocol had to be developed.

```go
type message struct {
        Opcode, SocketId    int
        Result, Query, Hash string
        Problems            []problem.Problem
        Difficulty          problem.Difficulty
}
```

The client application is initiated by visiting the servers web-page which download the client application[11]. Upon requesting the application web-page, the server spawns a goroutine[12] handling the http response writing task to deliver the application to the client. When the client application initiates it will open a keep-alive web-socket connection to the server where communication can occur. If the client wants the server to perform its service it will submit a request with operation code 0, which means that the client is requesting a problem to solve in order to acquire the servers trust. The server will generate a set of puzzles tied to the specific client. To prevent an attacker from reusing solutions for multiple requests or forging problem sets the problem set ϕ_i is chosen by the server and stored in memory. The server will reply with the set of puzzles and operation code 1, telling the client to solve the problem set. The client will solve the puzzles, using a brute-force approach since no better algorithm is known. The client will submit the solutions with operation code 1. Upon receiving a request with operation code 1, the server will verify the solution. If the verification succeeds the server will grant the client some cpu time, otherwise the connection will be terminated.

[9]But the expressiveness, clarity and performance of go programs is not to be dismissed.
[10]HTML, CSS, Javascript and all that magic that make stuff happen in the browser
[11]The client application is built in javascript which is distributed to the client via a web-page.
[12]Goroutines are golangs built in multi-threading primitive.

4 Simulation Experiments

The reputation based PoW system's architecture and load management mechanism was tested by conducting a series of simulated experiments using custom made web-based population-simulation interface. The configuration parameters of the experiments are described in detail in section 4.1.

In the experiments, a fixed set of legitimate users was programmed to send requests to a single server. To simulate a legitimate population each client was programmed to send the requests with stochastic based delays and thereby simulating the unpredictable rate of requests. Each request was also programmed to perform a fixed load on the server side to simulate the execution of a service. As a result a normal work load on the server was simulated.

```
for i := int64(0); i < 250000000; i++ {
        //simulate some server load (~80 ms)
}
```

The requests of legitimate users was mixed with a larger population of adversaries to mount a DoS attack. The adversaries was programmed to have lesser to zero delay in-between requests, thus forcing the server to service more requests than a legitimate user.

A custom made web-based monitor was used to aid the understanding well the RB-PoW system performed under different scenarios. The monitor uses real-time graphs to outline information about CPU utilization, requests rates and the time to solve PoW puzzles. In each run we studie the service time for adversaries, legitimate and mobile users to see how the protection system affected each type of behaviour. Control parameters such as number of attacking connections and settings for delays was changed between experimental runs, as-well as the parameters for server's internal reputation mechanism.

A logging web service was developed. The client sends data containing response, solving and granting time as well as at tag describing if the data came from a mobile, legitimate desktop user or from an adversary, for every request made to the server, to the logging service. The datalogger compile the data to human readable format, enabling us to examine results and compute statistics.

The goal of the experiments was to examine how well the reputation based PoW architecture performed in mitigating DoS attacks and at the same time serving legitimate users and users with lesser hardware. The degree of effectiveness from the experiments conducted was determined by following observations:

- A comparison of how many attacking connections that was required to launch DoS attacks with comparable levels of performance degradation, with and without reputation based PoW.

- How much legitimate users and users with lesser hardware was affected by DoS attacks, with and without reputation based PoW protection.

4.1 Setup

The simulation experiments resulted in benchmarking the Reputation Based Proof of Work with the initially proposed Proof of Work protocol. In this section, the set-up of the benchmarks will be presented.

4.1.1 The legitimate users

The legitimate users had to be fixed. By being consequent, a fair comparison could be done simulating the same normal work load on the server through all the simulations. The benchmarks was run with 100 legitimate users, each being set to having a normally distributed access times expected to ten seconds with a fifteen second standard deviation. Consequently, the most probable delays would vary between between zero to forty seconds.

4.1.2 Mobile user

The mobile users where set to have the same behaviour as the legitimate users in respect to request rate. Due to a relative lack of testing hardware the cellphones clients where only four by number, running on a single device.

4.1.3 The attackers

The simulation experiment will test the solution for two different kinds of malicious behaviour, both seeking to drain the server resources.

Flooding Behaviour: The attackers try to submit as many requests as possible, solving the problems as fast as possible regardless if proof-of-work is activated. The test set-up consists of seventeen Quad Core Q9550 2.83Ghz machines running Ubuntu operative system each handling four client processes in parallel.

Draining Behaviour: The attackers try to have many connections open in parallel, even if it means an increased number of contexts switches and longer solving time for each problem when proof-of-work is activated. This set-up consist of twelve equally specified machines each running 40 clients in parallel.

The server load is in fact only affected by the total request rate, but the reason to test both flooding as well as draining is to examine performance of RB-PoW under different circumstances, since a lower individual request rate among adversaries may well disguise concerned adversary amidst legitimate users.

5 Results

We performed two sets of experiments to design to test and study the effectiveness of RB-PoW in comparison to classical PoW;

- *Server Flooding Attacks:* in this experiment, the server was flooded with adversaries programmed to have no delays between requests.

- *Server Draining Attacks*: in this experiment, the server was drained by having a large amount of adversaries faking to be legitimate users by having identical delays as the legitemate users.

5.1 Mitigation against Server Flooding

Table 5.1. Results from simulated server flooding. All time measurements in milliseconds (ms). Attacker population size is seventeen computers simulating four clients each. The results are the mean of sample data with a 99% confidence level that the population mean is within the interval.

Prot. model	**Attackers**			**Legitimate users**		**Mobile devices**	
	Pop. size	Solving	Service	Solving	Service	Solving	Service
PoW	17x4	1266±9	1794±12	3129±98	3652±99	31959±4576	32488±4556
RB-PoW	17x4	1875±43	2312±43	267±13	582±22	2975±1031	3469±1103

5.2 Mitigation against Server Draining

Table 5.2. Results from simulated server draining. All time measurements in milliseconds (ms). Attacker population size is twelve computers simulating fourty clients each. The results are the mean of sample data with a 99% confidence level that the population mean is within the interval.

Prot. model	**Attackers**			**Legitimate users**		**Mobile devices**	
	Pop. size	Solving	Service	Solving	Service	Solving	Service
PoW	12x40	12700±199	14377±210	2616±119	3707±145	59119±14150	61005±14175
RB-PoW	12x40	14615±380	15371±383	4555±332	5206±345	29940±13426	30753±13490

6 Conclusions

In this paper, we have explored the possibilities of a reputation based proof-of-work protocol in mitigating DoS attacks. We have presented the RB-PoW protocol, an experimental architecture and tested the protocol with a web-based simulation interface. We presented results of two simulation experiments, server flooding and server draining with two types of implementations; classical proof-of-work and the proposed reputation based proof-of-work. The results are the average milliseconds of our simulation data with a confidence level of 99%, see Appendix 1 for detailed explanation in the significance of our results. Furthermore, the results from these simulation experiments will be brought to discussion in following paragraphs with the goal of answering the initial problem statements:

6.1 Server Flooding

The results of simulating server flooding, see table 5.1, shows that the RB-PoW protocol is a vast improvement to mitigating DoS flooding attacks in comparison the the classical PoW. It is interesting to note that the RB-PoW perform better in all three user-type cases. The most significant improvement was to the mobile users where the service time was reduced approximately 9 folds.

This finding was unexpected and suggests that PoW in its classical implementation is very primitive. There are several possible explanations for this result and the most significant reason is likely to be the fact that the classical PoW does not in any way separate legitimate users from adversaries. There are, however, other possible explanations. It may in fact be the result of inadequate tuning of the PoW protocol.

6.2 Server Draining

Contrary to expectations, the results of simulating server draining in table 5.2 show that classical PoW is slightly better at servicing legitimate users with a service time approximately 1.4 times better than RB-PoW. However, RB-PoW still performs significantly better than classical PoW regarding mobile users, with a almost 2 times faster service time. Although, the performance of servicing mobile users in both protocols would in a real world context be far from acceptable.

Even though the results was quite contradictory to our expectations the explanation for this result is rather reasonable. The proposed RB-PoW system is based on differentiating malicious behaviour from legitimate behaviour. However, in the simulation experiment the adversaries was programmed to fake themselves as legitimate users, thus defeating the core concept of RB-PoW.

6.3 Concluding Remarks and Feasibility

The present results are significant in two major aspects. First and foremost, the results show that existing proof-of-work protocols can be improved by using clever and adaptive scaling of problem difficulties. Reputation based scaling is a feasible approach in improving the dynamics of the proof-of-work concept and has shown to improve the taxing impact on adversaries while having a less effect on legitimate users during denial-of-service attacks with flooding characteristics.

Another result of significance is that the RB-PoW protocol during our simulation of draining type of denial-of-service attacks at worst performed like the cpu-scaling PoW implementation.

The findings are very valuable in the aspect that they show that RB-PoW could work as an effective substitute to current proof-of-work schemes, since it provides a vast improvement in protection against some attacks while at the same time does not seem to perform significantly worse in situations less than ideal to RB-PoW.

6.4 Lessons learned

During the our implementation of the RB-PoW protocol and the observations made during simulation experiments we have conducted:

- That moving costs of services onto the clients help in mitigating denial-of-service attacks. The RB-PoW protocol show a potential way of maintaining server stability even under changing conditions.
- An adaptive implementation of proof-of-work schemes effectively puts high impact on adversaries and low impact on legitimate users during server flooding.
- Tuning of server controllable parameters and choice of reputation system have very strong effects of effectiveness regarding RB-PoW protocols. The mathematical model of the reputation system is a major parameter when scaling problem difficulties based on the behaviour of the users.
- The protocol could be improved by using bit-strings instead of byte-strings to make the difficulty levels more fine grained.
- A non-linear difficulty model could potentially be more effective against large scale flooding type of denial-of-service attacks.

6.5 Suggested Directions for Future Research

The investigation presented in this paper is experimental in its nature. For further research we suggest the following two possible extensions of the current protocol.

6.5.1 Non-linear difficulty scaling

Green et al. in "Reconstructing hash reversal based proof of work schemes" demonstrates that hash-reversal schemes based server load would be ineffective when under attack by GPU utilizing adversaries. On the contrary it was shown that, schemes which adapts accordingly to client behaviour could be effective against such an attack. Therefore a direction for further research would be to explore the possibilities of improving our proposed protocol with a non-linear difficulty scaling as improvement to cope with potential GPU-based DoS attacks.

6.5.2 IP-address spoofing mitigation

The RB-PoW model may be susceptible to ip-address spoofing attacks, where an attacker tries to appear as many different clients with low computational capabilities, if the adversary is able to read packages sent to other ip-addresses. This scenario is plausible if an adversary Alice is on the same network as Bob providing the service, or if Alice has access to some large subnet which she can use to fake clients. Explore methods to mitigate this type of attack.

Appendices

1 Statistical Confidence

An important question of our study is, *when do we trust our data?* The results presented in Section 5 was collected from the simulation experiments run through the web-based simulation interface. The data was collected as samples during a certain time-frame during the experiments, dividing data into the categories of *adversaries*, *legitimate users* and *mobile device users*. However, can we be certain that the calculated averages of our samples represent the average of the whole population?

The answer lies in statistical mathematics. To bring confidence in the presented results it is important that a average of the sample data can, with a certain probability, be found within a interval of confidence, also known as confidence interval. Hence, knowing this interval enables the possibility to infer that the average of one sample is significantly different from another, as long as the confidence interval of the averages do not overlap.

One fundamental requirement of finding these confidence intervals is that the distribution of the samples is known. However, the distribution of our the result data in our simulation experiments is likely a sum of pascal and unknown distributed variables. Because of the uncertainty a bit of magic is required to solve this problem.

1.1 Arithmetic Averages Have a Bit of Magic

That bit of magic is the *Central Limit Theorem*. The theorem states that, given a sufficiently large sample of identically distributed independent variables,[13], each with a well-defined mean and well-defined variance[14], will be approximately normally distributed[18].

From this theorem two implications can be drawn that is relevant for our test data:

- A random sample scan be taken from any population, in our case samples of the simulation experiments, even if the simulation data is not normally distributed and assume it to be approximately normally distributed.

- The theorem also allows assumptions to be made about the sample data of the simulation regardless of the entire simulation data. Thereby, an interval estimation can be made about the true average of the simulation data with only sample data.

1.2 Confidence in Test Results

The probability that the average of whole population falls within the interval of the result data is either 1 or 0 - the interval captures the average or it doesn't[18].

[13]The central limit theorem generally takes effect when samples is larger than 30.

[14]This implies that both the mean the variance should be finite.

However, with a 99% confidence it is (probably) safe to assume that the average of the population actually falls within the confidence interval of the result data.

Bibliography

[1] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. “DOS-resistant authentication with client puzzles”. In: *Lecture Notes in Computer Science*. Springer-Verlag, 2000, pp. 170–177.

[2] Ari Juels and John G. Brainard. “Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks”. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA*. The Internet Society, 1999. ISBN: 1-891562-04-5. DOI: `http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/juels.pdf`.

[3] Ben Laurie and Richard Clayton. *“Proof of Work” proves not to work*. 2004.

[4] Jeff Green et al. “Reconstructing hash reversal based proof of work schemes”. In: *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*. LEET’11. Boston, MA: USENIX Association, 2011, pp. 10–10. URL: `http://dl.acm.org/citation.cfm?id=1972441.1972455`.

[5] Ravinder Shankesi, Omid Fatemieh, and Gunter A Carl. “Resource inflation threats to denial of service countermeasures”. In: UIUC Tech. Report (2010). URL: `http://hdl.handle.net/2142/17372`.

[6] Cynthia Dwork and Moni Naor. “Pricing via Processing or Combatting Junk Mail”. In: *Advances in Cryptology - CRYPTO 92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 139–147. ISBN: 3-540-57340-2. DOI: `http://link.springer.de/link/service/series/0558/bibs/0740/07400139.htm`.

[7] Adam Back. *Hashcash - A Denial of Service Counter-Measure*. Tech. rep. 2002.

[8] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: (2011). URL: `http://bitcoin.org/bitcoin.pdf`.

[9] D. Mankins et al. “Mitigating Distributed Denial of Service Attacks with Dynamic Resource Pricing”. In: *Proceedings of the 17th Annual Computer Security Applications Conference*. ACSAC ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 411–. ISBN: 0-7695-1405-7. URL: `http://dl.acm.org/citation.cfm?id=872016.872173`.

[10] Andrei Serjantov and Stephen Lewis. “Puzzles in P2P systems”. In: *In 8th CaberNet Radicals Workshop, Corsica.* 2003.

[11] Adam Back. *Hashcash benchmarks.* URL: `http://www.hashcash.org/benchmark/` (visited on 04/09/2013).

[12] Patrick P. Tsang and Sean W. Smith. “Combating spam and denial-of-service attacks with trusted puzzle solvers”. In: *Proceedings of the 4th international conference on Information security practice and experience.* ISPEC’08. Sydney, Australia: Springer-Verlag, 2008, pp. 188–202. ISBN: 3-540-79103-5, 978-3-540-79103-4. URL: `http://dl.acm.org/citation.cfm?id=1788494.1788508`.

[13] *Secure Hash Standard (SHS).* National Institute of Standards and Technology. 2012.

[14] NIST William E. Burr. *NIST COMMENTS ON CRYPTANALYTIC ATTACKS ON SHA-1.* URL: `http://csrc.nist.gov/groups/ST/hash/statement.html` (visited on 04/23/2013).

[15] Fredrik Henriques and Niklas Nordmark. *Proof of Work.* 2011.

[16] Gian-Paolo D. Musumeci and Mike Loukides. *System Performance Tuning, 2nd Edition (O’Reilly System Administration).* O’Reilly Media, Inc., 2002. ISBN: 059600284X.

[17] Google. *The Go Programming Language.* 2013. URL: `http://golang.org` (visited on 04/06/2013).

[18] G. Blom. *Sannolikhetsteori och statistikteori med tillämpningar.* Bok C. Studentlitt., 1975. ISBN: 9789144035925. URL: `http://books.google.se/books?id=H\_jMQQAACAAJ`.