Ana I. Pereira · Andrej Košir ·
Florbela P. Fernandes · Maria F. Pacheco ·
João P. Teixeira · Rui P. Lopes (Eds.)

Communications in Computer and Information Science 1754

Optimization, Learning Algorithms and Applications

Second International Conference, OL2A 2022
Póvoa de Varzim, Portugal, October 24–25, 2022
Proceedings

Springer

OL2A
INTERNATIONAL
CONFERENCE

Communications in Computer and Information Science 1754

Ana I. Pereira · Andrej Košir ·
Florbela P. Fernandes · Maria F. Pacheco ·
João P. Teixeira · Rui P. Lopes (Eds.)

Optimization, Learning Algorithms and Applications

Second International Conference, OL2A 2022
Póvoa de Varzim, Portugal, October 24–25, 2022
Proceedings

Springer

Editors
Ana I. Pereira
Instituto Politécnico de Bragança
Bragança, Portugal

Andrej Košir
University of Ljubljana
Ljubljana, Slovenia

Florbela P. Fernandes
Instituto Politécnico de Bragança
Bragança, Portugal

Maria F. Pacheco
Instituto Politécnico de Bragança
Bragança, Portugal

João P. Teixeira
Instituto Politécnico de Bragança
Bragança, Portugal

Rui P. Lopes
Instituto Politécnico de Bragança
Bragança, Portugal

ISSN 1865-0929 ISSN 1865-0937 (electronic)
Communications in Computer and Information Science
ISBN 978-3-031-23235-0 ISBN 978-3-031-23236-7 (eBook)
https://doi.org/10.1007/978-3-031-23236-7

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This CCIS volume 1754 contains the refereed proceedings of the Second International Conference on Optimization, Learning Algorithms and Applications (OL2A 2022), a hybrid event held during October 24–25, 2022.

OL2A 2022 provided a space for the research community on optimization and learning to get together and share the latest developments, trends, and techniques, as well as to develop new paths and collaborations. The conference had more than three hundred participants in an online and face-to-face environment throughout two days, discussing topics associated with optimization and learning, such as state-of-the-art applications related to multi-objective optimization, optimization for machine learning, robotics, health informatics, data analysis, optimization and learning under uncertainty, and Industry 4.0.

Five special sessions were organized under the following topics: Trends in Engineering Education, Optimization in Control Systems Design, Measurements with the Internet of Things, Advances and Optimization in Cyber-Physical Systems, and Computer Vision Based on Learning Algorithms. The OL2A 2022 program included presentations of 56 accepted papers. All papers were carefully reviewed and selected from 145 submissions in an single-blind process. All the reviews were carefully carried out by a scientific committee of 102 qualified researchers from 21 countries, with each submission receiving at least 3 reviews.

We would like to thank everyone who helped to make OL2A 2022 a success and hope that you enjoy reading this volume.

October 2022

Ana I. Pereira
Andrej Košir
Florbela P. Fernandes
Maria F. Pacheco
João P. Teixeira
Rui P. Lopes

Organization

General Chairs

Ana I. Pereira	Polytechnic Institute of Bragança, Portugal
Andrej Košir	University of Ljubljana, Slovenia

Program Committee Chairs

Florbela P. Fernandes	Polytechnic Institute of Bragança, Portugal
Maria F. Pacheco	Polytechnic Institute of Bragança, Portugal
João P. Teixeira	Polytechnic Institute of Bragança, Portugal
Rui P. Lopes	Polytechnic Institute of Bragança, Portugal

Special Session Chairs

João P. Coelho	Polytechnic Institute of Bragança, Portugal
Luca Oneto	University of Genoa, Italy

Technology Chairs

Alexandre Douplik	Ryerson University, Canada
Paulo Alves	Polytechnic Institute of Bragança, Portugal

Local Organizing Chairs

José Lima	Polytechnic Institute of Bragança, Portugal

Program Committee

Ana I. Pereira	Polytechnic Institute of Bragança, Portugal
Abeer Alsadoon	Charles Sturt University, Australia
Ala' Khalifeh	German Jordanian University, Jordan
Alexandre Douplik	Ryerson University, Canada
Ana Maria A. C. Rocha	University of Minho, Portugal
Ana Paula Teixeira	University of Trás-os-Montes and Alto Douro, Portugal
André Pinz Borges	Federal University of Technology – Paraná, Brazil
André R. da Cruz	Federal Center for Technological Education of Minas Gerais, Brazil

Andrej Košir	University of Ljubljana, Slovenia
Arnaldo Cândido Júnior	Federal University of Technology – Paraná, Brazil
António J. Sánchez-Salmerón	Universitat Politécnica de Valencia, Spain
António Valente	Universidade de Trás-Os-Montes e Alto Douro, Portugal
Bilal Ahmad	University of Warwick, UK
Bruno Bispo	Federal University of Santa Catarina, Brazil
Carlos Henrique Alves	CEFET/RJ, Brazil
Carmen Galé	University of Zaragoza, Spain
B. Rajesh Kanna	Vellore Institute of Technology, India
Carolina Gil Marcelino	Federal University of Rio de Janeiro, Brazil
Christopher E. Izquierdo	University of Laguna, Spain
C. Sweetlin Hemalatha	Vellore Institute of Technology, India
Damir Vrančić	Jozef Stefan Institute, Slovenia
Daiva Petkeviciute	Kaunas University of Technology, Lithuania
Dhiah Abou-Tair	German Jordanian University, Jordan
Diego Brandão	CEFET/RJ, Brazil
Dimitris Glotsos	University of West Attica, Greece
Diamantino Silva Freitas	University of Porto, Portugal
Eduardo Vinicius Kuhn	Federal University of Technology – Paraná, Brazil
Elizabeth Fialho Wanner	Federal Center for Technological Education of Minas Gerais, Brazil
Elaine Mosconi	Université de Sherbrooke, Canada
Esteban Clua	Federal Fluminense University, Brazil
Eric Rogers	University of Southampton, UK
Felipe N. Martins	Hanze University of Applied Sciences, Netherlands
Florbela P. Fernandes	Polytechnic Institute of Bragança, Portugal
Florentino F. Riverola	University of Vigo, Spain
Gaukhar Muratova	Dulaty University, Kazakhstan
Gediminas Daukšys	Kauno Technikos Kolegija, Lithuania
Glaucia Maria Bressan	Federal University of Technology – Paraná, Brazil
Glotsos Dimitris	University of West Attica, Greece
Humberto Rocha	University of Coimbra, Portugal
J. Marcos Moreno Veja	University of Laguna, Spain
João Paulo Carmo	University of São Paulo, Brazil
João P. Teixeira	Polytechnic Institute of Bragança, Portugal
Jorge Igual	Universidad Politécnica de Valencia, Spain
José Boaventura-Cunha	University of Trás-os-Montes and Alto Douro, Portugal
José Lima	Polytechnic Institute of Bragança, Portugal

Joseane Pontes	Federal University of Technology – Ponta Grossa, Brazil
Juan Alberto G. Esteban	Universidad de Salamanca, Spain
Juan A. Méndez Pérez	University of Laguna, Spain
Juani Lopéz Redondo	University of Almeria, Spain
Julio Cesar Nievola	Pontifícia Universidade Católica do Paraná, Brazil
João Paulo Coelho	Polytechnic Institute of Bragança, Portugal
Jorge Ribeiro	Polytechnic Institute of Viana do Castelo, Portugal
José Ramos	NOVA University Lisbon, Portugal
Kristina Sutiene	Kaunas University of Technology, Lithuania
Lidia Sánchez	University of León, Spain
Lino Costa	University of Minho, Portugal
Luis A. De Santa-Eulalia	Université de Sherbrooke, Canada
Luís Coelho	Polytecnhic Institute of Porto, Portugal
Luca Oneto	University of Genoa, Italy
Luca Spalazzi	Marche Polytechnical University, Italy
Maria F. Pacheco	Polytechnic Institute of Bragança, Portugal
Manuel Castejón Limas	University of León, Spain
Marc Jungers	Université de Lorraine, France
Marco A. S. Teixeira	Universidade Tecnológica Federal do Paraná, Brazil
Maria do R. de Pinho	University of Porto, Portugal
Marco A. Wehrmeister	Federal University of Technology – Paraná, Brazil
Markus Vincze	TU Wien, Austria
Martin Hering-Bertram	Hochschule Bremen, Germany
Mikulas Huba	Slovak University of Technology in Bratislava, Slovakia
Michał Podpora	Opole University of Technology, Poland
Miguel Ángel Prada	University of León, Spain
Nicolae Cleju	Technical University of Iasi, Romania
Paulo Lopes dos Santos	University of Porto, Portugal
Paulo Alves	Polytechnic Institute of Bragança, Portugal
Paulo Leitão	Polytechnic Institute of Bragança, Portugal
Paulo Moura Oliveira	University of Trás-os-Montes and Alto Douro, Portugal
Pavel Pakshin	Nizhny Novgorod State Technical University, Russia
Pedro Luiz de P. Filho	Federal University – Paraná, Brazil
Pedro Miguel Rodrigues	Catholic University of Portugal, Portugal
Pedro Morais	Polytechnic Institute of Cávado e Ave, Portugal
Pedro Pinto	Polytechnic Institute of Viana do Castelo, Portugal

Roberto M. de Souza	Federal University of Technology – Paraná, Brazil
Rui P. Lopes	Polytechnic Institute of Bragança, Portugal
Sabrina Šuman	Polytechnic of Rijeka, Croatia
Sani Rutz da Silva	Federal University of Technology – Paraná, Brazil
Santiago Torres Álvarez	University of Laguna, Spain
Sara Paiva	Polytechnic Institute of Viana do Castelo, Portugal
Shridhar Devamane	Global Academy of Technology, India
Sofia Rodrigues	Polytechnic Institute of Viana do Castelo, Portugal
Sławomir Stępień	Poznan University of Technology, Poland
Teresa P. Perdicoulis	University of Trás-os-Montes and Alto Douro, Portugal
Toma Roncevic	University of Split, Croatia
Uta Bohnebeck	Hochschule Bremen, Germany
Valeriana Naranjo-Ornedo	Universidad Politécnica de Valencia, Spain
Vivian Cremer Kalempa	Universidade Estadual de Santa Catarina, Brazil
Vitor Duarte dos Santos	NOVA University Lisbon, Portugal
Wynand Alkema	Hanze University of Applied Sciences, Netherlands
Wojciech Paszke	University of Zielona Gora, Poland
Wojciech Giernacki	Poznan University of Technology, Poland
Wolfgang Kastner	TU Wien, Austria

Contents

Machine and Deep Learning

Management of Virtual Environments with Emphasis on Security

André Mendes[1,2,3(✉)]

[1] Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal
a.chaves@ipb.pt

[2] Laboratório para a Sustentabilidade e Tecnologia em Regiões de Montanha (SusTEC), Instituto Politécnico de Bragança, Campus de Santa Apolónia, 5300-253 Bragança, Portugal

[3] Department of Systems Engineering and Automatics, Universidade de Vigo, 36.310 Vigo, Spain

Abstract. With the popularity of using virtual environments, it urges important measures that increase its security, as well as maintain a good user experience. A widespread attack is a denial of service which proposes to break the availability of service through a large number of illegitimate requests employing all computing resources of the target and degrading the user experience. In order to be effective in this particular type of attack, usually powerful equipment or a combination of them is required. This article proposes a new approach to this attack through a language-based Erlang application, which uses the processing power of a low-cost device. Its use would open the possibility of effective attacks coming from devices with less processing power, or from IoT devices, but capable of at least degrading the experience of a legitimate user, anonymously.

Keywords: Virtual environments · Attacks of denial of service · Infrastructure · Parallel computing

1 Introduction

The adoption of virtual environments has become more and more common. The emergence of cloud computing has not only enriched the diversity of computing platforms but also promoted the rapid development of big data, the Internet of Things (IoT) and other fields and its use gives organisations greater flexibility for on-demand usage. Cloud computing is defined as a type of parallel and distributed system, consisting of a set of virtualised and interconnected computers, thus enabling its dynamic availability. Three service modes, namely, infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS), are provided through cloud computing, allowing enterprises to use resources in accordance with their current needs [8]. Finally, it is presented the

A. I. Pereira et al. (Eds.): OL2A 2022, CCIS 1754, pp. 93–106, 2022.
https://doi.org/10.1007/978-3-031-23236-7_7

computational resources in a unified way and benchmarked according to the level of service (SLA) [1].

In this way, companies can avoid the need to build and maintain new platforms. Envisioning this growth, technology giants such as Amazon, Google and Microsoft have created and offered proprietary solutions, however, open solutions also exist. Due to the unique advantages of the dynamic, virtual and almost infinite expansion offered by cloud computing, an increasing number of scientific applications and business services are being migrated from traditional computing platforms to cloud computing platforms.

One of the possibilities that virtualisation offers is the reconfiguration of virtualised resources, conferring *levels of elasticity*. That is, resources in a given physical machine can be increased, decreased or migrated to other machines as required by the administrator. For cloud service providers, the use of virtualisation technology enables the integration of scattered and heterogeneous physical resources, which not only improves the utilization and maintenance level of hardware equipment but also mitigates various adverse effects caused by hardware failure [6].

Despite being an excellent solution, virtual environments have some threats that can compromise the operation and credibility of services, for example, loss of data, performance drop, unavailability of the service, in addition to the challenge of maintaining the correct elasticity of the infrastructure.

In order to determine some of the limitations of virtualised environments, this study aims to warn about the use of new technologies demonstrating a new form of denial of service attack, where with the help of parallel computing, it is possible to make unavailable large services through the use of a simple tool.

Therefore, this work aims to collaborate in order to assist in the configuration of virtual environments in a more secure way, to make efficient use of computational resources and, finally, to maintain a more reliable and secure high availability environment.

The paper is organised as follows. Section 2 brings some related works and the 3 section demonstrates the main tool used. Section 4 describes the system model. The results of the overload tests are presented and discussed in Sect. 5, and finally, Sect. 6 concludes the work and lists future works.

2 Related Works

Virtual machine (VM) allocation algorithms can be divided into *static* and *dynamic* strategies [4]. In *static* allocation, a scheduling strategy is generated for VMs according to general task information and resource requirements. In simple terms, a static algorithm provides a solution for the initial placement of VMs over the minimum number of active physical machines (PMs) to maximise the energy efficiency and resource utilization of the data centre. However, such an algorithm does not provide a solution for the reallocation of VMs to new PMs considering dynamic workloads.

In [10] the authors established a mathematical model that considers the mutual influence between VMs to reduce resource contention. Thus, the optimal

scheduling of VMs can be calculated under a given amount of energy allocation. In [5], a proposed hierarchical resource management solution that considers the correlations between VMs and the interconnection network topology of these VMs to maximize the overall benefit.

In contrast to static allocation, *dynamic* VM allocation optimises power consumption and cost when migrating existing VMs or forecasting future tasks. During the VM allocation process, new workloads can be accepted by the data centre; as a result, removing some PMs or adding a new PM can make the solution sub-optimal.

Sandpiper [15] is a system that can gather information about the utilisation of PMs and VMs. By finding overloaded PMs, the resource usage of PMs can be handled. Tasks in VMs with a sizeable volume-to-size ratio (VSR) are pushed to VMs with smaller values.

In [16], the authors proposed an approach employing the long-term memory encoder-decoder network (LSTM) with an alertness mechanism. This approach draws the sequential and contextual features of the historical workload data through the encoder-decoder network and integrates the alertness mechanism into the decoder network.

In [9], the authors proposed a VM allocation strategy, which dynamically redistributes VMs based on the actual needs of a single VM, taken as a reference. The proposed solution considers different types of resources and is designed to minimise underuse and overuse in cloud data centres. The experiments highlighted the importance of considering multiple types of resources. However, the above methods do not take into account the communication cost between VMs, which would affect the power consumption in the VM scheduling problem.

In [3], the authors considered the scenario of communication failures that block data transmission. To improve the reliability of data transmission, they improved a Greedy Perimeter Stateless Routing (GPSR) scheme based on perimeter positioning. The work described in [2] proposed a dynamic method of reconfiguration to solve the problem of instability and high resource occupancy of traditional approaches. The LSTM neural network was trained using a web service QoS dataset and service summoner information collected from sensors. The candidate service datasets from the cloud and edge service were used as input and predicted variables.

In [7], several papers present virtual machine migration approaches with advantages such as load balancing, fault tolerance and also energy savings.

None of these works addresses a limitation of virtualised environments, which makes it possible, with the use of parallel computing, to degrade the quality of large services by using a simple tool, based on the Erlang language, that uses the processing power of a low-cost machine, such as an IoT device.

3 Tsung Tool

Tsung is an open-source multi-protocol distributed load testing tool [11]. Currently, the application can be used to stress servers with the following protocols: HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP, MQTT and XMPP.

It has been developed for the purpose of simulating users in order to test the scalability and performance of IP-based client-server-type applications. Moreover, existing protocols can be extended by changing the code and adapting to the new reality.

The choice of using Tsung was mainly based on its capacity to simulate a large number of simultaneous users from a single machine. Also, because of the possibility of distributing the load in a cluster of several client machines in order to simulate hundreds of thousands of virtual users in the same test.

The programming language Erlang [14] was chosen to implement Tsung for being a concurrency-oriented language and for its ability to create real-time, lightweight and massively scalable systems. The tool was created with 3 main features in mind: performance – capable of supporting hundreds of thousands of lightweight processes in a single virtual machine; scalability – with a naturally distributed runtime environment, promoting an idea of process location transparency; and fault tolerance – built to develop robust and fault-tolerant systems, e.g. bad responses received by the server do not interrupt the measurement during test execution.

Besides complete documentation for its use, this application has several features, such as a configuration system based on *XML*, the possibility of creating dynamic sessions with the storage of answers from the server in runtime for later use, monitoring of the Operating System, processing, memory and network traffic through agents in remote servers, the ability to generate realistic traffic and the creation of detailed reports of the tests performed.

To set up the tests, an *XML* file is used with tags that determine all the necessary aspects for its realization. All the scenarios are described between the labels *Tsung* where the registered degree is determined, which stands out: emergency, critical, errors, warnings and debugging. The higher the registered degree, the higher the impact on the test performance.

The first configuration to be done is the determination of the IP address of the client machine and the server to be tested. The server is the entry point to the cluster, in case several servers are added weight is defined for each one (by default it's 1) and each session will choose a server randomly.

Besides, the port number and a protocol type are configured for the server, which may be TCP, UDP, SSL (IPv4 and IPv6) or WebSocket (Listing 1.1).

```
<clients>                                                    1
    <client host="localhost" use_controller_vm="true"/>      2
</clients>                                                   3
<servers>                                                    4
    <server host="server1" port="80" type="tcp" weight="4"/> 5
    <server host="server2" port="80" type="udp" weight="4"/> 6
</servers>                                                   7
```

Listing 1.1. Server and client configuration.

After that, the progression of the test is defined through phases. Each phase has a duration and the number of users to be created. The test ends when all the users finish their sessions. In the following example (Listing 1.2) there are 3 phases with 2 min duration each. In the first, 2 new users are created every

second, in the second 10 new users are created every second and, finally, in the last, 1000 new users are created per second.

```
<load>
    <arrivalphase phase="1" duration="2" unit="minute">
        <users interarrival="0.001" unit="second"/>
    </arrivalphase>
    <arrivalphase phase="2" duration="2" unit="minute">
        <users interarrival="0.00001" unit="second"/>
    </arrivalphase>
    <arrivalphase phase="3" duration="2" unit="minute">
        <users interarrival="0.000001" unit="second"/>
    </arrivalphase>
</load>
```

Listing 1.2. Configuration of test phases.

Finally, the user sessions are defined where the requests to be executed are described (Listing 1.3). Each session can receive a probability or a weight that is used to determine which session will be chosen for the user to be created. There is the possibility to set an exponentially distributed wait time between requests to better simulate a legitimate request. Another option is to read variables in files and use them at runtime in the created requests.

```
<sessions>
    <session name="session-1" probability="50" type="ts_http">
        <request>
            <http url="/" method="GET" version="1.1"></http> </
request>
            <request> <http url="/images/logo.gif" method="GET"
version="1.1" if_modified_since="Tue, 14 Jun 2022 02:43:31 GMT">
            </http>
        </request>
        <thinktime value="20" random="true"></thinktime>
        <transaction name="req-index">
            <request> <http url="/index.en.html" method="GET"
version="1.1" >
            </http> </request>
            <request><http url="/images/header.gif" method="GET"
version="1.1">
            </http> </request>
        </transaction>
        <thinktime value="60" random="true"></thinktime>
            <request>
                <http url="/" method="POST" version="1.1"
contents="search=test">
                </http>
            </request>
    </session>
        <session name ="next-session" probability="50" type="
ts_psql" > . . .
    </session>
</sessions>
```

Listing 1.3. Configuration of sessions.

In Listing 1.3 there is an example with two sessions, the first with the name "session-1" will occur half the time, because it has a probability of 50%, being of type HTTP. Initially, two GET requests are made, one from the root home page and another from the "logo" image if it was modified after June 14, 2022. After that, a random pause of exponential distribution with an average of 20 s is made and then a transaction named "index_request" is created, which is composed of

two GET requests. There is another pause centred in 60 s, and, finally, a POST request passing the value "test" to the parameter *search*. The other session, of type *PostgreSQL*, will occur the other half of the time, and its requests have been omitted.

4 System Model

In order to perform denial of service tests, the following structure was set up: 4 VMs with 4 GB RAM, 4 Core Processor – similar to a Raspberry Pi 4 specs – and 128 GB Storage each, acting as *hosts*; 1 virtual switch with 1 Gbps ports interconnecting these *hosts*.

In one of the hosts, the Xen project hypervisor was installed and created an internal VM with a web server, thus simulating a service provider environment. The other 3 hosts have the Debian Linux operating system installed and were used either to simulate traffic or to perform a denial of service attack.

When creating the environment to perform the tests, the first step was to verify the limits within which the application could work. That is, to verify the level of depletion that the attacker host could support in order to mark the work to be developed. Therefore, as the first tests aimed to monitor and set limits, tests were performed with a high load of generation users.

After that, the web server was stressed with a large number of requests for a long time in order to follow its behaviour. Random times between requests were introduced in order to resemble real requests.

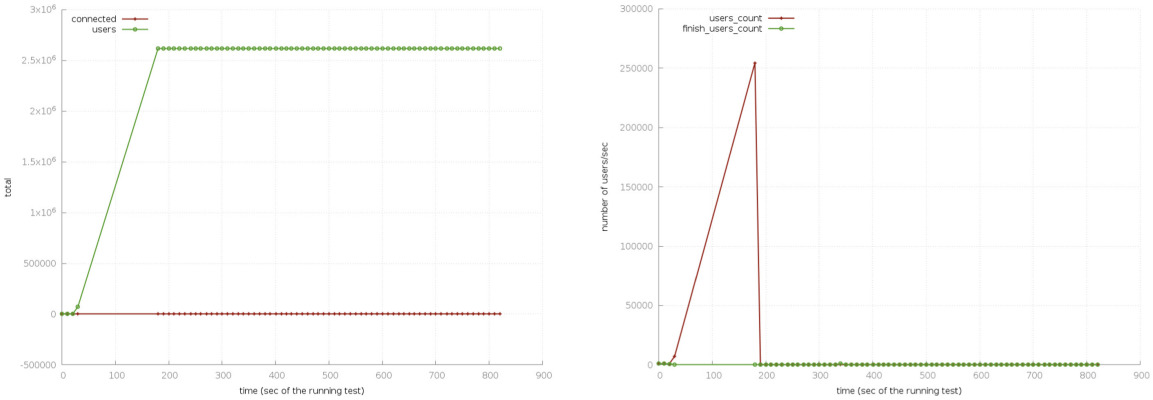
Finally, a traditional attack and the performed attack were compared, highlighting their differences and similarities having as reference *3 factors*: consumed bandwidth, processing load and amount of free memory. With the *overload* of any of these 3, it is possible to generate a denial of service or, at least, the degradation of the user experience, delaying the response time for a legitimate request.

5 Results

Through **first test**, the set of parameters that limit the usage of the attacking machine through maximum user generation, at maximum performance, was sought. The test was divided into 3 phases, where in the initial 30 s, 1.000 users per second were generated; in the following 120 s, 10.000 users per second were generated; and in the last 30 s, 100.000 users per second were generated.

Thus, each user made a connection to the web server and performed an HTTP GET request of the file *index.html*. After receiving any response from the server, the client ceased to exist.

Analyzing the results (Fig. 1), one can observe that after a certain time (approximately 180 s), there was no more generation of users. Besides, it was noticed that the web application stops handling the requests because the growth



(a) Simultaneous. (b) Arrival rate.

Fig. 1. Amount of users – $1^{\underline{st}}$ Test.

of concurrent users is constant. It can also be verified that the application supported the creation of up to 12.500 users per second, but for values above that there was no response.

Next, the more extensive **second test**, where in the initial 30 min, 100 users per second were generated; in the following 273 min (4,56 h), 1.000 users per second were generated; and, in the last 30 min, 10.000 users per second were generated.

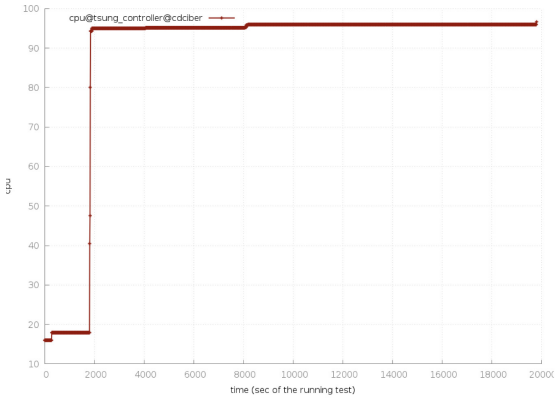
It can be seen that the second phase of the test (after 1.800 s) demands the entire processing load from its beginning until almost 20.000 s (Figs. 2a and 2b). Considering that the second phase lasted until 18.180 s, it is safe to say that the application, under these conditions, has a limit to the creation of 1.000 users per second (Fig. 2c).

The application behaved well when treating 100 users per second in the first phase – until 1.800 s (Fig. 2c). However, in the following phase, there was an instability where, in spite of the first requests having been handled, as more requests arrived, there was a significant overload, maintaining an average of 500 requests per second being served, following this way until the end of the test. This information is confirmed by the processor graphic showing that there was no alteration with the beginning of the third phase, from 8.000 s on (Fig. 2b).

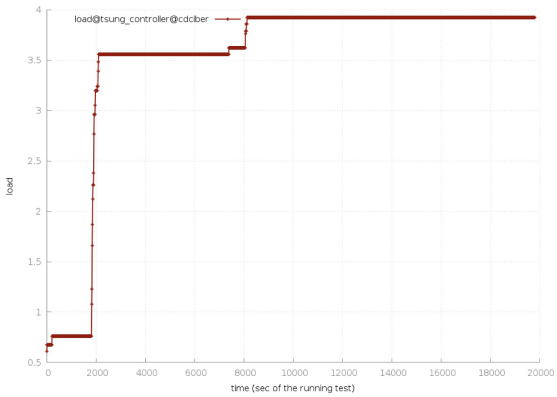
Analyzing the report of this test in Table 1, it could note that from the total of users created, only 8.45 million users were treated, that is, they received as a response the requested page. It is worth pointing out that the average request waiting time was 426 s. Another observation was that the connection with the longest wait was 1.034 s.

After these initial tests to determine the setup parameters of the application, the behaviour of a denial of service (DoS) attack using the *flooding technique* was studied and verified, as it is one of the most common types of DoS.

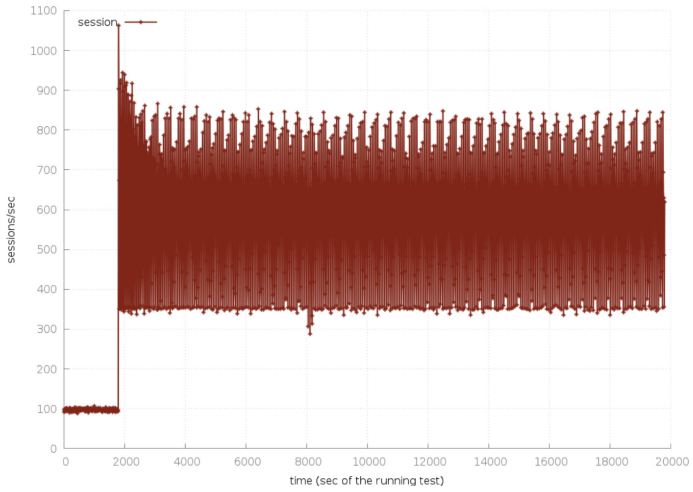
Traffic flooding occurs in connections using the Transmission Control Protocol (TCP), utilised in services that require reliable data delivery, by exploiting



(a) CPU usage (%).



(b) CPU load.



(c) Sessions per second rate.

Fig. 2. 2^{nd} Test.

the connection opening process of this protocol, which begins with the negotiation of certain parameters between the client and the server.

TCP was designed specifically to provide a reliable end-to-end data stream on an untrusted inter-network. An inter-network differs from a single network because its various parts may have completely different topologies, bandwidths, delays, packet sizes, and other parameters [13].

In a standard TCP connection, the client sends a synchronisation packet (SYN) to the server, requesting to open the connection. Upon receipt of the SYN packet, the server processes the connection request and allocates memory to store client information. Next, a synchronisation/confirmation packet (SYN/ACK) is sent as a reply to the client, notifying it that its connection request has been accepted. The client then sends an ACK packet to complete the connection opening. This procedure is known as a *three-way handshake* (Fig. 3c).

The flood attack consists of sending several SYN packets to the target, causing it to reserve a space in memory for the connection it will open. Following the flow, the target returns an SYN/ACK and waits for an ACK to close the connection, but the attacker sends another SYN request forcing the target to

Table 1. Report table – $2^{\underline{nd}}$ Test.

Name	Highest rate (users/s)	Average rate (users/s)	Total sessions
Connection	1.034,6	426,8	8.445.969
Page	1.034,6	426,79	8.445.969
Request	1.034.,6	426,79	8.445.969
Session	1.063,4	555,17	10.987.007

reserve another area of memory for a new connection. By performing this action on a large scale the attacker is able to exhaust the target's memory.

To analyze in more detail how each resource is exploited, some parameters related to the target and the attacker machines are set. Let t_a be the time interval between each TCP SYN segment sent by the attacker, t_p be the time required for the target to process a TCP connection request and send a response, and t_m be the time a given memory resource is allocated for a TCP connection.

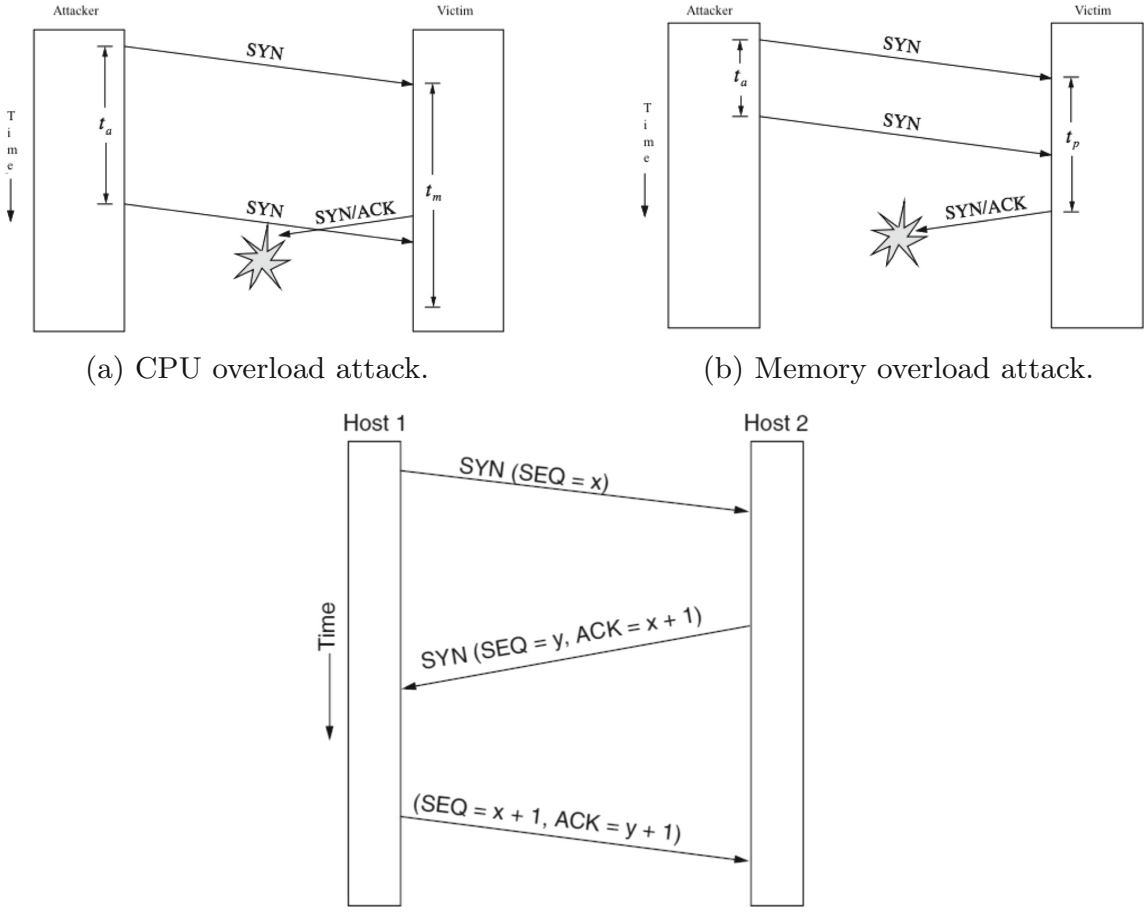
An *attack of CPU overload* on the target machine can be performed when the attacker can generate segments at a faster rate than the target machine can process them, that is when $t_a < t_p$, as shown in Fig. 3a. In this case, the target machine cannot process connection requests in a timely manner, which causes the request queue to fill up and many of them to be dropped.

Thus, should a legitimate user attempt to access the service under attack, it is very likely that their connection request will be dropped along with the attack traffic. This is because the legitimate user's traffic must compete for the same resource with the numerous segments sent by the attacker.

Importantly, to remain anonymous, the attacker does not need to use your real IP address to carry out the attack. In fact, any IP address can be used as the source address in attack packets. Using forged source addresses does nothing to change the effect suffered by the target machine. However, the response to the connection request does not return to the attacker, but rather to the IP address used in his packets. In this way, the attacker is able not only to deny the target service but also to remain anonymous.

Another resource that can be exploited during TCP SYN segment flood attacks is the target machine's *memory*. When the server receives a request to open a TCP connection, it allocates a small amount of memory to store certain information about the state of the connection, such as the initial sequence numbers, the IP addresses, and the TCP ports used for this connection. Figure 3b illustrates the case where the attacked resource is the memory of the target machine. It is important to note that, even with memory limits controlled by the operating system, denial of service occurs when the memory allocated for connections is exhausted, without the complete exhaustion of the victim's entire memory needed.

Next, a TCP SYN/ACK segment is sent back to the client in order to finalize the three-way agreement. At this point, the TCP connection is said to be half open. The initially reserved resources are then allocated until the connection



(a) CPU overload attack.

(b) Memory overload attack.

(c) 3-way handshake model [13].

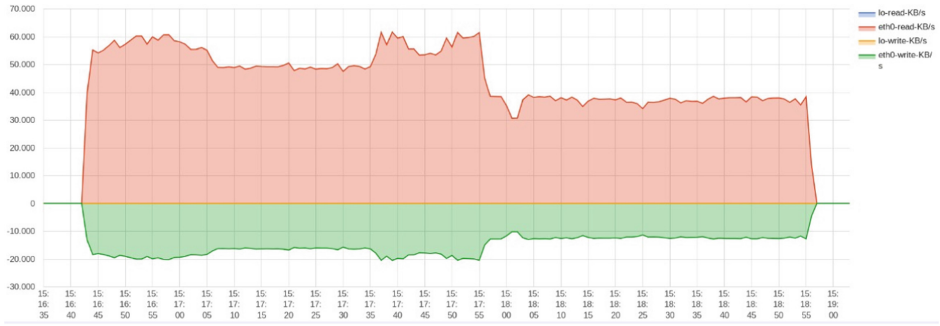
Fig. 3. TCP big picture.

is terminated by conventional means, or until a timer bursts indicating that the expected TCP ACK segment was not received. In this case, the allocated memory is finally freed to be used for future connections.

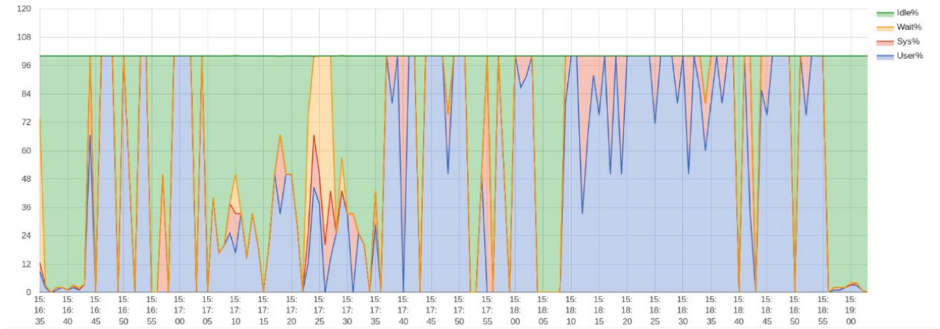
Note, by definition, that the inequality $t_m > t_p$ is always valid. Thus, *3 possibilities* can happen. The *first* possibility occurs when $t_a < t_p < t_m$, that is, the interval between sending attack packets is less than the victim's processing time. This is the case discussed earlier where the victim's processing is overloaded. It is also worth noting that in addition to processing, the victim's memory is also overloaded. This occurs because the connection requests that the victim manages to fulfil in time, allocate a memory space that is only released when the timer bursts, since packets with forged source addresses are used.

The *second* possibility occurs when $t_p \leq t_a < t_m$. In this case, the victim is able to process requests, but memory spaces are consumed by each new connection and are not freed up in time to service all connections. Therefore, new connections are not accepted due to a lack of memory.

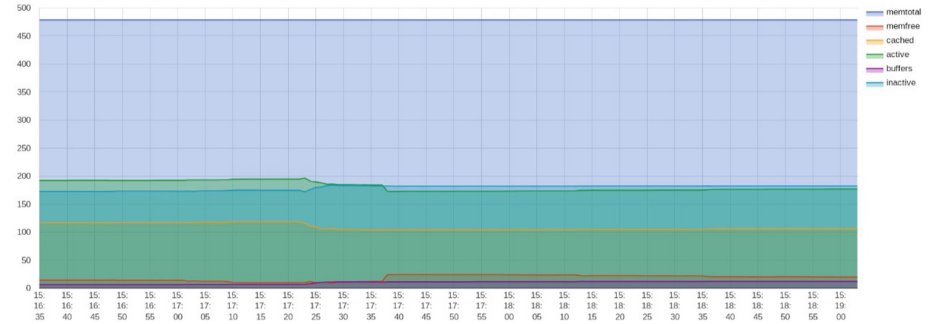
The *final* case occurs when the attacker cannot generate attack packets at a rate sufficient even to overflow memory, i.e., $t_a \geq t_m$. In this case, the victim's resources are not attacked directly and other forms of attacks are required to deny the offered service.



(a) Data traffic.



(b) CPU usage.



(c) Memory usage.

Fig. 4. Flood attack, *Hping3* tool as a reference – $3^{\underline{rd}}$ Test.

After that, let's move on to the DoS attack configuration, in which the *Hping3* [12] tool was used with the following configuration:

```
# hping3 -c 1000000 -d 120 -S -w 64 -p 80 --flood <IP>
```

where $-c$ refers to the number of packets to be sent; $-d$ refers to the size of each packet to be sent; $-S$ defines the sending of only SYN packets; $-w$ defines the size of the TCP window; $-p$ defines the TCP port to be attacked; and, $--flood$ parameter that makes the application send the packets as fast as possible, not caring about the response.

Next, the attack on the web server – **third** test – was performed using the *nmon* monitoring tool to collect data, through which large data traffic (sent/received), high processing load and CPU usage could be verified during the whole testing slot. And finally, RAM exhaustion of the target machine (Fig. 4).

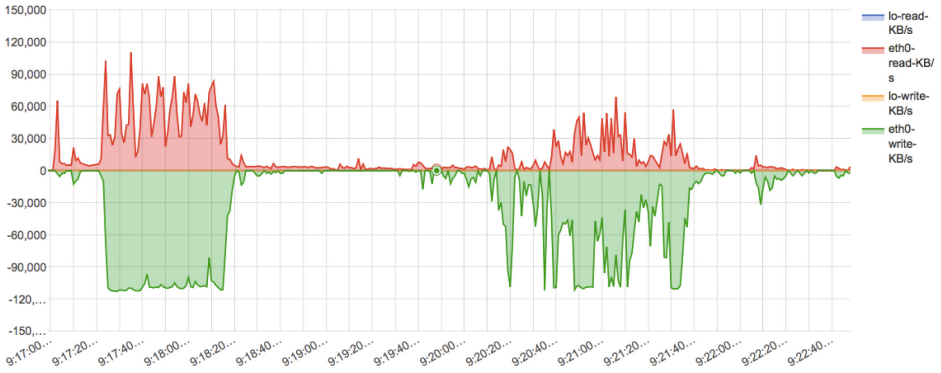
These results were employed as a reference to be used in a comparison of the tests performed with the *Tsung* application.

With the intention of verifying the target's behaviour, comparing the attack with Tsung and Hping3, the **fourth test** was prepared in which each user makes a TCP connection with the target machine and sends a 380 kB file, in 3 distinct moments, interspersed with a request of the root web page (*index.html*), the request of another web page with 2 MB and a random amount of time paused modelled by an exponential distribution with the average equal to 4 s (Listing 1.4).

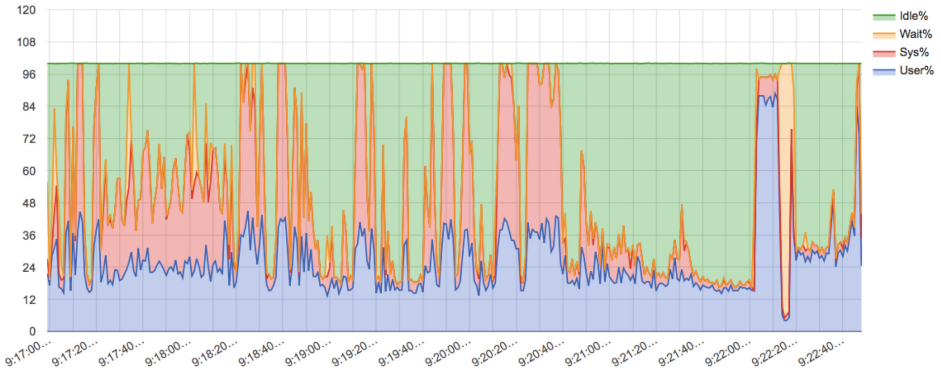
```
<http url="/" method="POST" contents_from_file="<path>/fake_data" />
<http url="/" method="GET"/>
<http url="/" method="POST" contents_from_file="<path>/fake_data" />
<thinktime value="4" random="true"></thinktime>
<http url="/great_tsung.html" method="GET"/>
<http url="/" method="POST" contents_from_file="<path>/fake_data" />
```

Listing 1.4. DoS attack configuration.

In this fourth test, Fig. 5, in the initial 10 s, 200 users per second were generated; in the following 40 s, 335 users per second were generated; in the next 40 s, approximately 500 users per second were generated; and in the last 20 s, 660 users per second were generated.



(a) Data traffic (kBps).



(b) CPU usage.

Fig. 5. Flood attack, *Tsung* tool – $4^{\underline{th}}$ Test.

It is possible to notice that all the available network bandwidth was occupied by the requests, generating some periods of high processing in the target machine. After all these checks, it was possible to create a test that resembles the flood test with the *Hping3* tool, which, besides occupying all the available bandwidth is capable of generating a high processing power in the chosen target machine.

With an infinite number of possibilities, both the memory, processing and network bandwidth of the target can be exhausted by changing the parameters in the *Tsung* tool, leaving it up to the attacker to configure the behaviour of his attack to approximate that of legitimate requests.

6 Conclusions

The *Tsung* tool has great flexibility in terms of configuration, allowing a high range of ways of use. As it is an application created in Erlang, a concurrency-oriented language, this tool has the characteristic of being light and massively scalable. It was created to be a stress testing tool for other applications, this work explored its use to perform denial of service (DoS) attacks.

As demonstrated, it is possible to perform a massive amount of requests using a piece of a low-cost device (for instance, a Raspberry Pi) in the totality of its processing, consuming in great part the resources of the attacked machine and degrading the experience of the legitimate user, effectively causing the denial of service to a server.

Finally, it was possible to create a large number of users, each performing different types of requests, sending and receiving data from the target. Besides the possibility of making random pauses which can be used to hinder the detection and blocking of requests, besides keeping the connections open for a longer time, approaching legitimate requests.

Acknowledgements. This work has been conducted under the project "BIOMA – Bioeconomy integrated solutions for the mobilization of the Agri-food market" (POCI-01-0247-FEDER-046112), by "BIOMA" Consortium, and financed by the European Regional Development Fund (FEDER), through the Incentive System to Research and Technological Development, within the Portugal2020 Competitiveness and Internationalization Operational Program.

The authors are grateful to the Foundation for Science and Technology (FCT, Portugal) for financial support through national funds FCT/MCTES (PIDDAC) to CeDRI (UIDB/05757/2020 and UIDP/05757/2020) and SusTEC (LA/P/0007/2021).

References

1. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Futur. Gener. Comput. Syst. **25**(6), 599–616 (2009)
2. Gao, H., Huang, W., Duan, Y.: The cloud-edge-based dynamic reconfiguration to service workflow for mobile ecommerce environments: a QoS prediction perspective. ACM Trans. Internet Technol. (TOIT) **21**(1), 1–23 (2021)

3. Gao, H., Liu, C., Li, Y., Yang, X.: V2VR: reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability. IEEE Trans. Intell. Transp. Syst. **22**(6), 3533–3546 (2020)
4. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The cost of a cloud: research problems in data center networks (2008)
5. Hwang, I., Pedram, M.: Hierarchical, portfolio theory-based virtual machine consolidation in a compute cloud. IEEE Trans. Serv. Comput. **11**(1), 63–77 (2016)
6. Jain, R., Paul, S.: Network virtualization and software defined networking for cloud computing: a survey. IEEE Commun. Mag. **51**(11), 24–31 (2013)
7. Le, T.: A survey of live virtual machine migration techniques. Computer Sci. Rev. **38**, 100304 (2020)
8. Mell, P., Grance, T., et al.: The nist definition of cloud computing (2011)
9. Mosa, A., Sakellariou, R.: Dynamic virtual machine placement considering CPU and memory resource requirements. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 196–198. IEEE (2019)
10. Nasim, R., Taheri, J., Kassler, A.J.: Optimizing virtual machine consolidation in virtualized datacenters using resource sensitivity. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 168–175. IEEE (2016)
11. Niclausse, N.: Tsung documentation (2000)
12. Sanfilippo, S., et al.: Hping-active network security tool (2008)
13. Tanenbaum, A.S., Feamster, N., Wetherall, D.J.: Computer Networks, 6th edn. Pearson Education India (2020)
14. Virding, R., Wikström, C., Williams, M.: Concurrent Programming in ERLANG. Prentice Hall International (UK) Ltd. (1996)
15. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Sandpiper: black-box and gray-box resource management for virtual machines. Comput. Netw. **53**(17), 2923–2938 (2009)
16. Zhu, Y., Zhang, W., Chen, Y., Gao, H.: A novel approach to workload prediction using attention-based lstm encoder-decoder network in cloud environment. EURASIP J. Wirel. Commun. Netw. **2019**(1), 1–18 (2019)